INCREMENTAL DATA MINING FOR ACTIVE AND ADAPTIVE KNOWLEDGE BASE FOR PATIENT IMAGE RETRIEVAL

By

Lin Lin

A Thesis Submitted to the Faculty of the

BIOMEDICAL ENGINEERING PROGRAM

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2001

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

_____ _____

Olivia L. Sheng
McCoy-Rogers Professor

Date

ACKNOWLEDGMENTS

I wish to express my appreciation to the following individuals for their assistance and support.

To the members of my thesis committee: Dr. Olivia Sheng, my advisor, provided me with tireless encouragement and guidance enabling me to expand my perceptions of medical informatics beyond traditional database systems to tackle new and exciting frontiers. Dr. Stuart Williams, who not only supported me strongly throughout my master research, but also provided me precious advice about how to do research and how to plan my career . Dr. Terry Ovitt, with his exuberance for radiology research, helped me understand how radiologists work and offered precious expert domain knowledge that is critical for this dissertation.

To Kevin McNeill and Kris Maloney, who made it possible to use the radiology research facility enabling me to learn how the radiology information system works in the University Medical Center (UMC).

Last but not least, to the Whitaker Foundation for the fellowship. Without its support all my research and study can't come true.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The general perception that the use of information technology (IT) in health care is 10 to 15 years behind that in other industrial sectors such as banking, manufacturing and airline is rapidly changing. Faced with an unprecedented era of competition and managed care, health providers are now exploring the opportunities for using IT to improve quality while simultaneously reduce the cost of health care.

Clinical decision support systems and expert systems (CDSSs / ESs) focus on utilizing artificial intelligence and data mining techniques to provide fast decision support for physicians. Although several success stories about CDSSs / ESs have been reported [Freudenheim 92, Nash 94], these systems usually lack the ability to adapt to pattern changes that are embedded in new data. This is due to the fact that the traditional algorithms utilized by these systems cannot learn on an incremental basis, i.e., once they are built, they cannot adjust their structures in which the knowledge is imbedded.

Lack of incremental learning ability is not a unique phenomenon in health care expert systems. In fact, most of the machine learning algorithms developed to date are limited in their ability to adjust learned rules based on new, incoming data. In the Internet Age, when new data keep coming in at a high speed, this is a serious limitation for decision support systems.

The main objective of this dissertation is to develop a new incremental neural network technique in order to support decision support systems' adaptive needs. An

Incremental Neural Net (INN) algorithm that utilizes hidden layer activations to incrementally learn new patterns from incoming data is proposed. We then applied it to the Image Retrieval Expert System (IRES), a clinical decision support system for radiologists in University Medical Center (UMC), University of Arizona. The performance comparison between the INN and traditional neural net approach are compared.

This chapter is organized as follows: section 1.1 briefly introduces the concept of data mining and incremental learning, which serve as technical foundations for this dissertation. Section 1.2 introduces the background of IRES project and describes its adaptive need. Section 1.3 addresses research motivation and objectives. Section 1.4 provides an overview of this dissertation.

1. 1 Introduction to Incremental Data Mining

Data mining, with its ability to extract interesting knowledge from large amount of data, is used extensively in industrial applications such as fraud detection, stock prediction, and customer relationship management. In the health care domain, data mining also has the its potential in prediction and decision support tasks. In this section, we will briefly introduce data mining and some of its limitations.

1. 1. 1 Data Mining: An Overview

Some times referred to as *knowledge discover in databases (KDD),* data mining is the process of "nontrivial extraction of implicit, previously unknown and potentially useful information from data in databases" [Piatetsky 91] (A detailed discussion about data mining's definition can be found in Chapter 2). It borrows techniques and methodologies from researches in several other fields including AI, machine learning, database and statistics.

To handle the increasing amount of data generated every day, many large-scale databases have been created for scientific research, government administration, healthcare as well as many other application areas. For example, patients' information is stored in several databases in UMC, University of Arizona. This explosive growth in data volume has generated an urgent need for data mining techniques, which, consequently, has become a research area with increasing importance [Fayyad 96].

There are many different classification schemes for data mining. It can be classified based on the type of databases it works on, the type of knowledge to be mined, or the type of technique utilized during the mining. Among these classification schemes, *type of knowledge to be mined* is the most popular one. The IRES project falls into classification rules category from this perspective. A detailed discussion of the taxonomy of data mining approaches will be covered in Chapter 2.

1. 1. 2 Incremental Data Mining

Traditional machine learning techniques frequently deal with stand-alone systems and small size, flat file format data. On the other hand, data mining handles online, operational systems with huge amount of data. This difference brings many new challenges to data mining. For example, it must handle different format of data; it needs to have efficient and scalable algorithms; it also should be able to mine information from different resources.

One new challenge for data mining is the ability to *incrementally* learn. In a real world setting, new data keep coming into an online operational database system. The addition of new data may cause two changes to the characteristics of the overall data: quantitative change or qualitative change. *Quantitative change* refers to the change of data size without subtle changes in the knowledge patterns imbedded in the data. This type of change requires scalability of data mining algorithms. Many researches have been done in this area. *Qualitative change* refers to the changes of the hidden pattern, underlying knowledge, or overall distribution of the data. To adapt to this type of change requires active data mining techniques. Unfortunately, traditional data mining methods can not deal with dynamic changes: they perform well when encounter new problems with similar patterns, but either "forget" the old patterns or need to be retrained from scratch when new patterns appear. Figure 1.1 shows the comparison of knowledge-base system, machine learning and incremental data mining systems with regard to their ability to adapt to change.

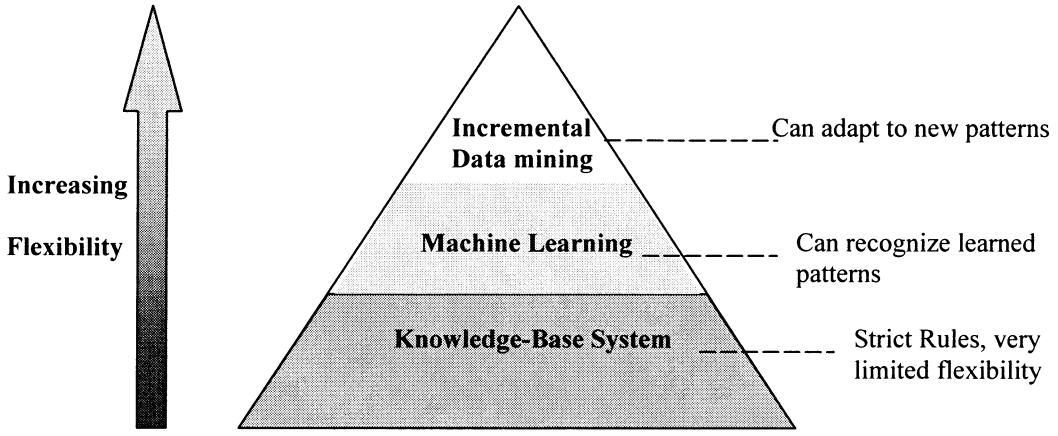


Figure 1.1 Comparison of Flexibility to Change among Learning Systems

Due to the difficulty of designing and implementing an incremental learning technique, past data mining researches have either assumed that the input data is static, or designed algorithms for arbitrary insertions and deletions of data records. However, to reach our goal of building active online data mining systems, the incremental learning barrier has to be broken. With the birth of E-commerce and Internet, incremental data mining is rapidly gaining attentions from academia as well as industry. This dissertation is an exploration work that develops, applies and evaluates an incremental data mining technique in a real world application, the IRES project.

1. 2 Background of Image Retrieval Expert System (IRES)

Advanced medical imaging technology has significantly expanded the role of radiology in clinical medicine. Practitioners have become increasingly dependent on information obtained from radiology examinations for clinical decision-making and patient management. In the meanwhile, advancements in information technology (IT)

have initiated and facilitated the progressive transformation of radiology from traditional film-based practices into a digital paradigm where radiologists can perform examination reading beyond organizational or geographical constraints. Jointly, the growing role of radiology and the recent trend toward a digital radiology practice have made patient image management a growing concern for many healthcare organizations. This dissertation reports a research effort that is dedicated to a more efficient and effective patient image management system.

1. 2. 1 Knowledge Base for Patient Image Retrieval

A crucial patient image management issue is image retrieval support to radiologists' examination reading. When performing primary reading on a newly taken examination, a radiologist often needs to make references to some prior images of the same patient for confirmation or comparison purposes [Sheng 94a]. Such image reference support is a crucial aspect of radiologists' examination reading and, when not properly provided, can become a severe service bottleneck or quality constraint. To alleviate the time and physical demands on reading radiologists, many healthcare organizations have taken a pre-fetch strategy to meet radiologists' patient image reference needs. This strategy selects a set of patient images that are presumably relevant to a current examination reading task and makes them available to the radiologist in advance of an examination reading.

The heuristic nature of radiologists' prior image reference knowledge makes knowledge-based approach sound and appropriate for patient image reference support.

Several knowledge-based approaches to support patient image retrieval and pre-fetching have been explored [Levin 90, Sheng 94b & 98, Lodder 91, Wilson 94]. Previous works in our lab in cooperation with University Medical Center (UMC), University of Arizona has enable us to capture the radiologists' behaviors and come up with a knowledge-base for image reference support. This knowledge-based system was shown to support radiologists' patient prior image reference needs more accurately than the current practice adopted at the study site [Sheng 94b & 94c]. It also serves as the basis for the Artificial Intelligence (A. I.) and data mining research carried out in our lab as well as in this dissertation.

1. 2. 2 Image Retrieval Expert System (IRES) Project

Knowledge-based systems are not without their own set of problems. A major problem is the difficulty to identify the knowledge initially and maintain the engineered knowledge base afterward. Furthermore, the need for continuous updating and customizing of the patient image retrieval knowledge, especially when supporting a different group of radiologists within or outside institutional boundaries, demands a more efficient and effective approach.

In contrast to the problem of and costs involved in the knowledge engineer-driven approach, AI-based automated learning techniques may represent a more effective and possibly more economical alternative to patient image retrieval. Such techniques can reduce the time required to engineer knowledge by many orders of magnitude, and they also offer the ability to adapt to the dynamically evolving nature of radiologists' image

retrieval behavior to some extent. Developing a machine learning solution for a particular task requires determining input attributes and decision outcomes as well as managing the characteristics of the target application. Based on the knowledge base we built, out lab has derived a set of learning requirements [Sheng 94b & 94c]. We will discuss these requirements in detail in chapter 3.

Using these requirements, two algorithms based on salient automated learning paradigms, neural network and multiple decision trees, have been designed and implemented by previous students. Each technique addressed the unique characteristics of patient image retrieval knowledge. The results showed that knowledge derived from the automated learning methods can achieve effective image retrievals that are comparable to those based on a knowledge-engineer-driven approach. The Image Retrieval Expert System (IRES) was designed and built upon the knowledge base derived from the knowledge-engineer-driven approach and the machine learning algorithms. The goal of this dissertation is to further extend the IRES into an active and adaptive system that responds to radiologists' evolving image retrieval requirements.

1. 3 Research Motivation and Objectives

An adaptive decision support system that automatically alters the decision rules based on new data is important for many modern applications. IRES is a good example. Traditional data mining algorithms are not capable of dealing with incremental learning tasks required by adaptive decision support systems. Past work in IRES project have demonstrated that machine learning algorithms, specifically neural networks, could

effectively support the decision making process of radiologists. The logical next step is to develop machine learning algorithms that can incrementally learn, apply them to IRES application, and build an adaptive knowledge system.

The objective of this dissertation is to develop an incremental learning neural network algorithm that can readjust its learning structure when new data come in based on the old learning structure it has. The development of such an algorithm serves as our first step toward adaptive knowledge systems. Compared to the traditional approach where the old learning structure is thrown away and a new one is built from scratch, this new algorithm should significantly save training time. We also want to examine the performance of the new algorithm compare to the traditional approach by examining their accuracy. A more detailed "research question" section will be presented after necessary literature reviews in chapter 2.

1. 4 Chapter Outline

The organization of the remainder of this dissertation is as follows: The following chapter covers background and basic literatures in the field of data mining, neural networks, and incremental learning. Based on these theoretical foundations, the research questions are presented. Chapter 3 reviews the primary radiological image reading process and provides an overview of Image Retrieval Expert System (IRES). It also goes over previous works in IRES project. In Chapter 4, I propose an incremental neural network approach called Incremental Neural Network (INN) and provide theoretical analysis as well as implementation details of it. Chapter 5 presents empirical comparisons

of the INN approach and the traditional neural network approach using IRES as the problem domain. The last chapter summarizes the findings and suggests future research directions. Topics addressed in individual chapters as they relate to the research questions are illustrated in Figure 1. 2.

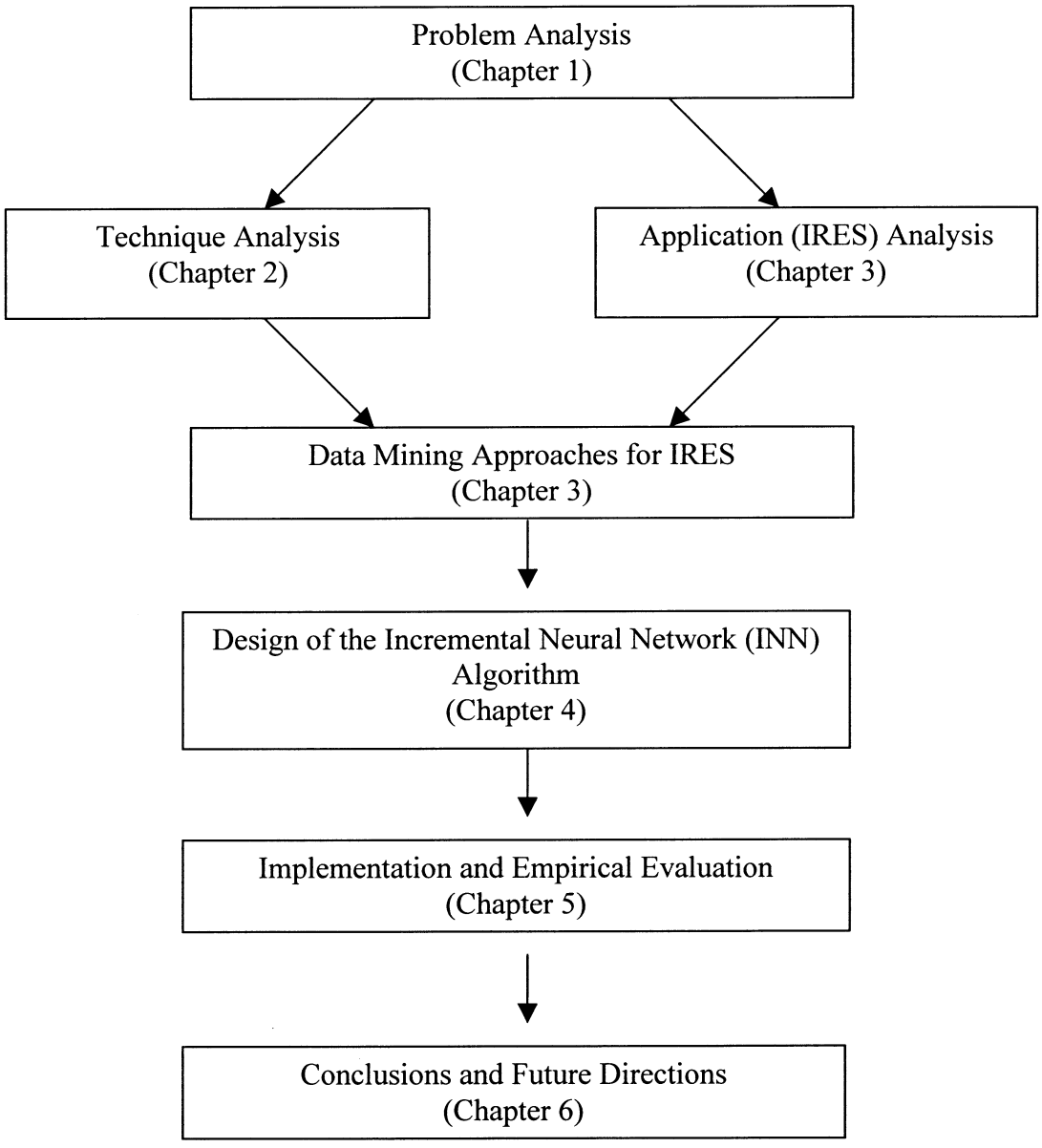


Figure 1. 2 Chapter Outline and Research Questions

CHAPTER 2

LITERATURE REVIEW AND RESEARCH FORMULATION

This chapter reviews the literature related to this dissertation research, focusing on two streams of research works: back propagation neural networks and incremental learning algorithms. Its aim is to identify common areas between these two issues and adequacies as well as inadequacies of existing approaches. Based on the literature review, the theoretical foundation of this dissertation research will be presented. Research question specific to the new algorithm developed in this research will also be detailed.

2.1 Literature Review on Data Mining

Before getting into discussion of specific neural network algorithms, we will first analyze the discipline of data mining in general. This will help us understand the context in which neural network techniques are developed, applied and modified.

2.1.1 Data Mining: Definition, process and taxonomy

Data mining is closely related to and frequently confused with another term: Knowledge Discovery in Database (KDD). One widely accepted definition about KDD is: *A non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data* [Piatetsky 91]. KDD contains the following steps:

1. Understand the application domain, the relevant knowledge, and the end-user goals

2. Create a target data set

3. Data Cleaning and preprocessing

4. Data reduction and projection: Reduce the effective number of variables under consideration to find the invariant representation of the data.

5. Choose data mining task

6. Choose data mining algorithms

7. Data mining.

8. Interpreting mined results and evaluation.

9. Consolidate discovered knowledge.

The whole process of KDD is visualized in Figure 2.1.

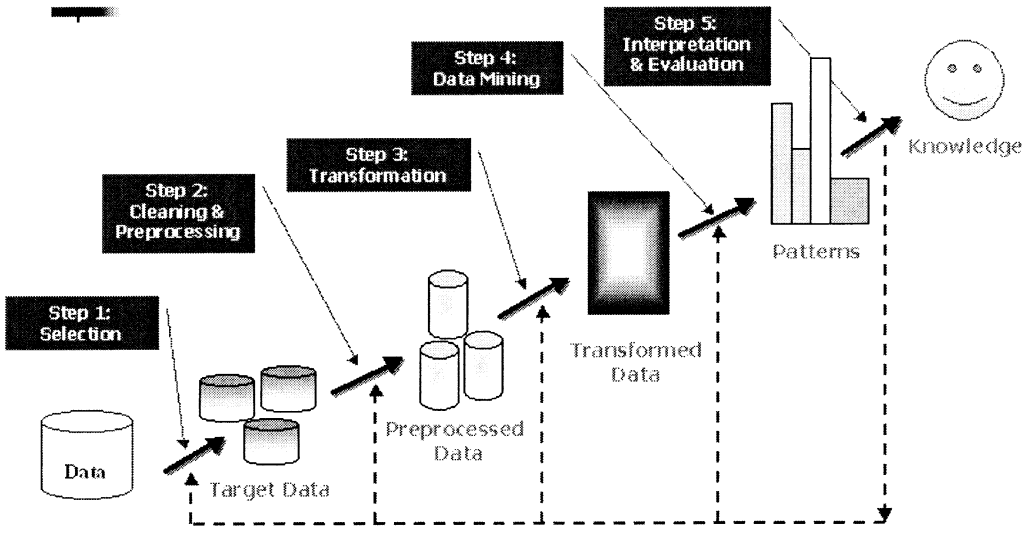


Figure 2. 1 Traditional KDD Process

According to this definition, data mining is one step in the KDD process consists of particular data mining algorithms that produces a particular enumeration of patterns over pre-processed data. Although there are several other definitions for data mining, and its definition is still controversial, this dissertation will adapt the following definition and deem data mining as one step in the whole knowledge discovery process: *Data mining is a process of nontrivial extraction of implicit, previously unknown and potentially useful information from data.* Based on this definition, I will directly use pre-processed data set to test the performance of the new algorithm in this thesis study. Data cleansing and transformation issues will not be viewed as part of data mining, and therefore will not be discussed.

Several categorization schemes that classify data mining into different types exit [Chen 96]. The following schemes are often used in the literature:

- Types of databases to work on: A data mining system can be classified according to the types of databases on which the data mining is performed. For example, a system is a relational-data miner if it works on a relational database, or an object-oriented one if it mines from object-oriented databases. Data mining can also be performed on simple flat file systems.

- Types of knowledge to be mined: Several types of knowledge can be discovered by data mining systems, including association rules, classification rules, evolution, time

series, clustering, and deviation analysis. This will be discussed in detail in the next section.

- Techniques to be utilized: Data mining can also be categorized according to the techniques it utilizes. There are three major approaches from this perspective: database management, statistical analysis and machine learning. *Database management* techniques provide the ability to extract from tables tuples (rows) that satisfy a common condition, but cannot determine what computations are worthwhile. *Statistical methods* have solid theoretical foundations, but the results of statistical analysis are often hard to interpret. Besides, many simplifying assumptions such as normal distribution are hard to maintain in reality. *Machine learning techniques* construct knowledge representations from data by using A.I. algorithms. They can deal with dynamic and noisy data, and are under extensive study.

The taxonomy of data mining is shown in Figure 2. 2. Among these and many other schemes, the *knowledge perspective* presents a clear picture on the nature of and requirement for data mining tasks. Therefore, we will continue our discussion about data mining using this scheme.

- ***Database Perspective:*** e.g., transactional databases, relational databases, O-O databases, flat file systems

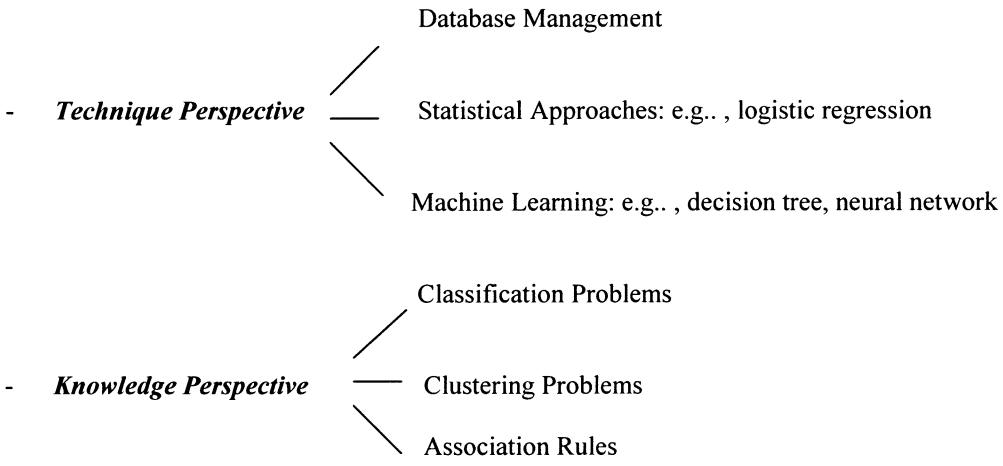


Figure 2. 2 Taxonomy of Data Mining

2.1.2 Three Classical Data Mining Tasks

Although real world applications differ from each other in nature, they generally contain three distinct types of knowledge that need to be extracted by data mining: Association rules, clustering, and classification.

- Association rules, also named "market basket analysis", can discover important associations among items such that the presence of an item will imply the presence of other items. It is the discovery of useful, novel links between two or more features (attributes) in a database. Such information is often very important in marketing and pattern recognition. A mathematical model [Agrawal 93] was proposed to address the

problem of mining association rules, and many algorithms have been proposed to solve this problem. Among them, Apriori [Agrawal 94] and DHP [Park 95] are two classic algorithms that have been used and modified by many researchers.

Association rules mining does not need to have historical data set as "teaching example". Instead, it directly processes the data and finds out hidden connections. This type of learning is named "*unsupervised learning*". Another type of learning, *supervised learning*, requires historical data as teaching examples in order to build a knowledge model.

- Clustering distributes data into different groups so that similar objects fall into the same group. It is also an unsupervised technique, and is sometimes called " *unsupervised classification*". Clustering analysis helps construct meaningful partitioning of a large set of objects based on a "divide and conquer" methodology that decomposes a large system into smaller components to simplify design and implementation. It has been studied in statistics [Jain 88], machine learning [Fisher 87], and data mining [Ester 95, Ng 94] with different emphases. In statistics, it mainly focuses on distance-based clustering analysis. In machine learning, clustering is offer referred to as "unsupervised classification", because which classes an object belongs to is not pre-specified. The distance measurement may not be based on geometric distance, but be based on certain conceptual classes. Clustering analysis in large databases and data mining has been studied recently in database community.

- Classification is the process of finding the common properties among a set of objects in a database and classifying them into different classes. The common properties of objects are often reflected by a series of attributes that describe their characteristics. To construct a classification model, a training data set from historical data is needed for an accurate mapping from the input space to the output space.

  Mathematically speaking, the training data set contains in it a mapping function $Y = f(X)$, where X is a vector representing input information (object properties), and Y is the actual outcome class an object with property X falls into. The goal of a classifier is to find a function $f'(W, X)$ that best simulates $f(X)$. W represents the parameters used by a classifier algorithm to adjust f' toward f.

  Classification is supervised learning. Applications of classification include medical diagnosis, performance prediction, selective marketing, etc.

  Data classification has been studied substantially in statistics, machine learning, neural networks, and expert systems [Weiss 91], and is an important theme in data mining [Fayyad 96].

When applying data mining to real world problems, we need to consider the type of knowledge we want to discover, and choose correspondent mining strategy. Figure 2.3 is a general guideline to choose best data mining technique based on nature of a problem.

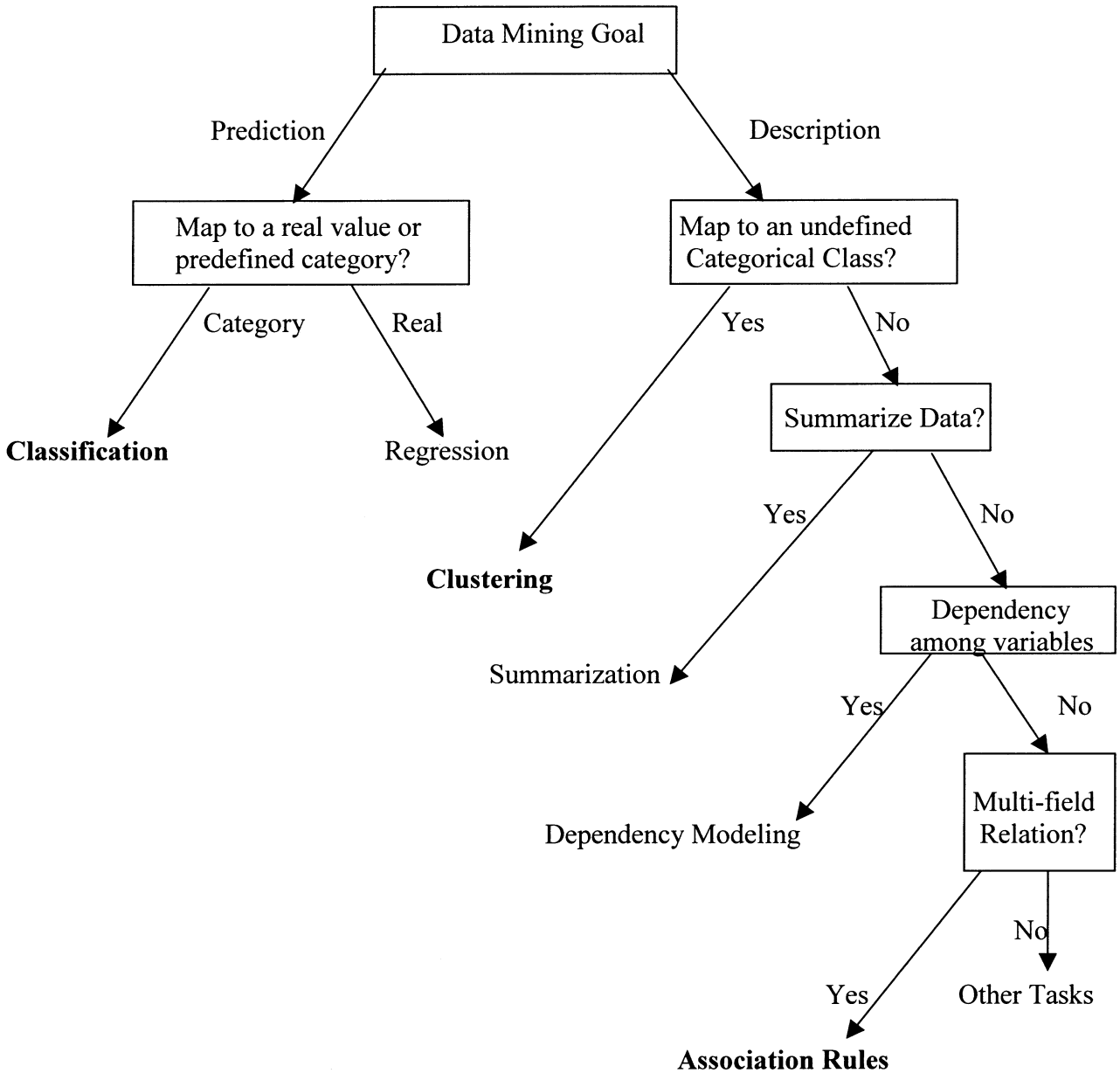


Figure 2. 3 choosing the right data mining task based on problem nature

We will see in next chapter that the nature of the IRES system makes it a classification problem. Therefore, we will concentrate on classification data mining in the rest of this dissertation.

In the past, researchers have proposed many classification models: neural networks, genetics algorithms, Bayesian methods, log-linear and other statistical methods, decision tables, and tree-structured models (so-called classification trees). Most of these techniques address the problem as follows: given a set of instances labeled as belonging to a particular class from historical data, determine a procedure to correctly assign new instances to these classes.

Since the theoretical basis for this research is in the field of neural networks, I will focus on literature review on this specific class of data mining algorithm in the next section.

2.2 Literature Review on Back Propagation Neural Networks

Artificial neural nets or simply "neural networks" go by many names such as connectionist models, parallel distributed processing models, and neuromorphic systems. In recent years, neural network research received extensive attention after decades of hibernation. With its powerful calculation ability and high performance in pattern recognition problems, neural net has become one of the major data mining techniques frequently used by industrial and academic researchers alike. In this section, we will look at the theoretical foundation as well as applications of neural networks.

2.2.1 Theoretical Foundation for Back Propagation Neural Networks

Neural network is a mathematical model that simulates the human brain. The basic unit in a neural network, an "artificial neuron", is a calculation unit that can get signals from many other artificial neurons as inputs and process them based on certain math functions to produce an output. The value of the output reflects the status of this neuron (firing or silent). The connections between neurons have adjustable weights. The simplest neuron sums up weighted inputs and passes the result through a nonlinearity function to produce output. An internal threshold is then used to judge whether the output is correspondent to firing or silent status. Figure 2.4 shows a simple neuron.

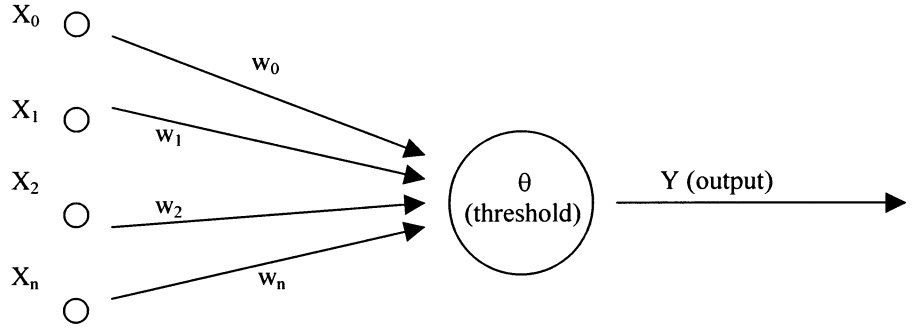


Figure 2.4 A simple neuron in neural network

Neural networks are specified by net topology (how neurons are connected to each other, how weight matrices are presented), node characteristics (internal threshold, nonlinearity functions) and training/learning rules (the function "f" in figure 2.4 as well as weight-adjust functions) [Lipp 87]. Different specifications represent different neural net algorithms such as Hopfield Net, Hamming Net, and Self-Organizing Maps.

Among the many different neural network algorithms, back propagation might be the most widely studied one. The whole model of back propagation is based on the theory of gradient descent. Since back propagation network serves as the mathematical foundation for the incremental neural network that is developed in this dissertation, it is necessary for us to go through some of its calculation details.

Figure 2.5 is a typical three – layer back propagation neural network. It is fully connected, which means each node in one layer is connected to all the nodes in the layer next to it. Each node in the input layer brings into the network the value of one independent variable. Each of the output layer nodes computes one dependent variable.

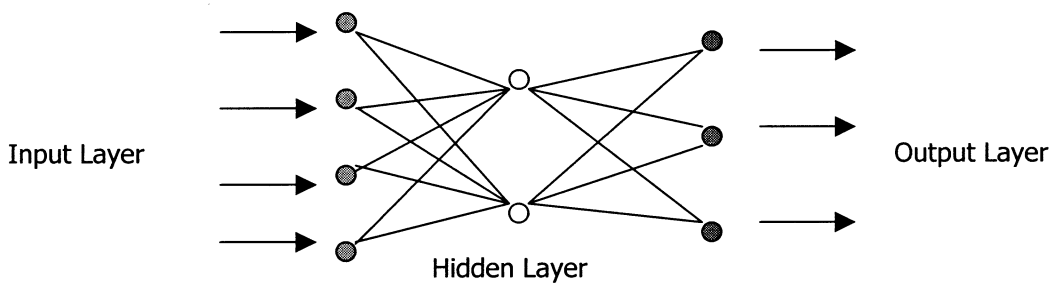


Figure 2. 5 A Three Layer Neural Network

The neural network operates in two modes: mapping mode and learning mode. In mapping mode, information flows forward through the network, from input layer to output layer. In learning mode, the information flow alternates between forward and backward. The whole training process is a combination of mapping mode and learning mode. We can describe the whole procedure in four steps:

First step is in mapping mode. At the beginning of the training, arbitrary values are assigned to the connection weights randomly. A set of input values is loaded onto the

input layer of the network. Each hidden node calculates the weighted sum of the inputs. Suppose we have M input nodes (from i_1 to i_M), J hidden layer nodes (from h_1 to h_J), and N output nodes (from O_1 to O_N). The weight from i_m to h_j is $W1_{mj}$, and the weight from h_j to On is $W2_{jn}$. The value for a hidden node h_j is therefore:

$$h_j = \sum_{m=1}^{M} i_m W1_{mj} \qquad (2.1)$$

Sometimes a bias weight may be added to above equation:

$$h_j = W1_{0j} + \sum_{m=1}^{M} i_m W1_{mj} \qquad (2.2)$$

The hidden layer nodes then undergo a "squash" operation so that their final values will all be in the range of [0, 1] or [-0.5, 0.5]. Usually a sigmoid function is used to achieve this:

$$h_j = \frac{1}{1+e^{-h_j}} \qquad (2.3)$$

Then, similar calculation will be carried out for the output layer:

$$O_n = \frac{1}{1+e^{-U_n}} \qquad (2.4)$$

Where Un is calculated as followed:

$$U_n = \sum_{j=1}^{J} h_j W2_{jn} \qquad (2.5)$$

The end product of step one is an estimated output based on the input value and the weight matrix values.

During step two, the output nodes are informed of the target (real) output values for the input example. The errors, i.e., the differences between the actual output values and the target output values, are calculated (T_n stands for the target value of node O_n):

$$\delta_n = O_n - T_n \qquad (2.6)$$

In step three, based on the difference between the actual output and the target output, each output node determines in which direction each of its weights would have to move to reduce the error. This step involves calculation of error derivatives. The equation for adjusting the output weights is:

$$\Delta W2_{jn} = \delta_n \bullet O_n \bullet (1 - O_n) \bullet h_j \qquad (2.7)$$

There are many different strategies for adjusting the weights, one of them is:

$$W2_{jn} = W2_{jn}' - \varepsilon \Delta W2_{jn} \qquad (2.8)$$

$W2_{jn}'$ stands for $W2_{jn}$ calculated in last training example. ε is called learning rate. Its value reflects how drastic the change of weight is for each learning step, and is very important in setting the speed of convergence of the network. Another factor, named momentum factor (μ), determines the effects of previous weight adjustments on the current ones. When properly chosen, this factor can accelerate network convergence, reduce potential oscillations, and enable the training to escape from local minimums [Weiss 91]. The equation that takes into account of the momentum factor is :

$$W2_{jn} = \mu W2_{jn}' - (1 - \mu)\, \varepsilon \Delta W2_{jn} \qquad (2.9)$$

Step four is similar to step three. The changes in the hidden layer weights are determined based on error derivative. The result of mathematical calculation gives the

following equation. Although it looks complicated, it is deduced based on the same gradient descent theory as step three:

$$\Delta W1_{mj} = \sum_{n=1}^{N} [W2_{jn} O_n (1 - O_n) \delta_n] \bullet h_j \bullet (1 - h_j) \bullet i_m \quad (2.10)$$

Again, different scheme may be applied to decide how this change should be applied to the weight. Traditionally, the schemes of weight adjust for the output weights and the hidden weights should be the same.

After these four steps, we have fed a case into the network, let it calculate the result and adjust its structure so that it has a better chance to correctly predict the outcome for this specific case next time. We will then repeat these four steps using many more examples. The training of a whole set of examples is called an epoch. Neural network training may run for many epochs before the performance of the network is good enough, i.e., its prediction is reliable.

After a neural network is trained, the input-to-hidden layer weight matrix captures the nature of the problem, and the hidden-to-output layer weight matrix "translate" this relationship back into output values.

An interesting theorem that shed some light on the capabilities of back propagation neural nets was proven by Kolmogorov and was described in [Irie 88, Hornik 89, Cybenko 89]. It stated that any continuous function of N variables can be computed using only linear summation and nonlinear but continuously increasing functions of only one variable. In another word, it effectively stated that a three layer neural network with N*(2N+1) nodes using continuously increasing nonlinearities can compute any

continuous function of N variables. This is a very strong statement, and it explains in part the generally good performance back propagation nets demonstrated.

2.2.2 Applications of Neural Networks

Besides their strong computational powers, back propagation nets also have the following characteristics [Luger 89]:

- They handle noise well. Once trained, neural networks show an ability to recognize patterns even though part of input data is missing or obscured.
- They are robust. Because the information is distributed, neural networks can survive the failure of some nodes (neurons).
- They can be applied to many different domains where other approaches have frustrated. Examples are speech recognition and image retrieval problems[Huang 93].

Above features enabled back propagation neural networks to become popular in real world applications. It was applied to bank failure predictions [Tam 92], speech synthesis and recognition [Sej 86], and visual pattern recognition [Rumel 86]. The IRES project also applied back propagation neural net for image retrieval pattern discovery [Sheng 98].

2.2.3 Limitation of Back Propagation Neural Networks

Although powerful in calculation, back propagation neural networks are not without their own weaknesses. Since the training of such networks often requires big

historical data sets to run through a fully connected graph, the time for training is often very long compare to other algorithms. Besides, tuning of the neural network (i.e. choosing values for parameters like learning rate, momentum, and hidden layer nodes number) has to be a trial-and-error approach, adding even more time to the whole training process. A detailed discussion of parameter tuning can be found in Chapter 5.

2.3 Literature Review on Incremental Learning

In the past, researchers dealt with static input data and stand-alone systems, and designed data mining solutions based on these systems' needs. With the introduction of Internet and the wide use of online databases, new problems such as scalability, heterogeneous databases and data integration occur. One major problem that most of the data mining techniques have to face today is the incremental learning problem. We will first examine the nature and cause of this problem, then review some past research works in this area.

2.3.1 Incremental Learning: Problem Definition

An important assumption for all supervised machine learning algorithms is that knowledge is embedded in historical data. The frequency that a certain pattern appears in the data set affects the strength of the corresponding rule learned. Take neural network for example. The method of deepest descent for error correction typically works with one pattern at a time [Widrow 90]. If the same pattern reappears in the historical data set for a

higher frequency, the weight matrices are tuned to fit that pattern better. As a result, the neural network will predict this pattern more accurately.

A natural conclusion from the above assumption is that for a *classifier* (i.e., a machine learning algorithms used for classification tasks), the ideal data set that will give it high performance should have similar patterns as the historical data set it is trained with. For application areas that are relatively consistent, this assumption stands correct. However, for more dynamic applications where new data with patterns never seen before coming in every day, the performance of the classifiers will face great challenge. When the new data has too many patterns different from the training data, we have to throw away the classifier trained before and rebuild a new one from scratch using the new data plus the old data as training data set in order to learn all the new patterns without "forgetting" the old ones. Apparently, this is an expensive operation.

An alternative is to "reuse" the trained classifier, since it already has old patterns embedded in it. By learning only the new patterns, we save precious training time. Because new data set is often much smaller than the original training data set, the amount of time saved might be significant.

Incremental learning tries to adaptively learn new knowledge embedded in incoming data by reusing the *knowledge structure* resulting from previous machine learning activities. By saving the time needed for a new round of machine learning from scratch, it can significantly improve the performance of an adaptive learning system.

In the above discussion, we frequently refer to the vague concept of "change" brought by new data. It is necessary to have a more detailed analysis of types of changes

that might occur. According to the nature of changes, I classified them into two major categories: knowledge infrastructure change and knowledge pattern change.

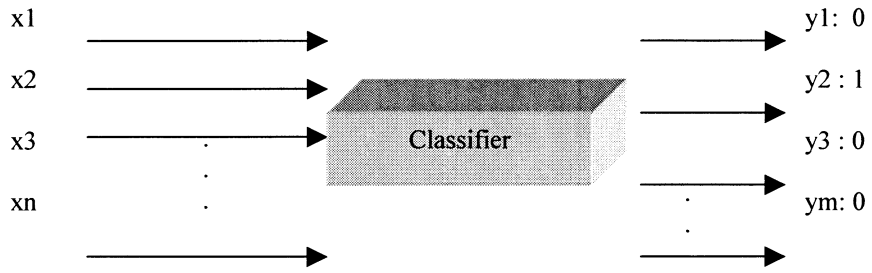


Figure 2.6 A Simple Classifier

Figure 2.6 describes a typical classification task. N input variables are fed into the classifier. After calculation, the classifier will produce the output. *Y1* to *Ym* are the m possible outcome classes an instance may fall into. A value "0" for yi means that the instance does NOT fall into the ith outcome category. In this figure, we see that the output falls into category *Y2*.

Knowledge infrastructure change refers to the situation where the addition of new input attributes or outcome classes occurs. Taking handwriting recognition as an example: Addition of one's handwriting data of number 10 into a database that only contains handwriting data of 1 to 9 will lead to a new outcome class "10" to be added to the problem. Input infrastructure may also be changed. For example, the availability of some input attributes may change, leading to the increase or decrease of dimensionality of the input space. In figure 2.6, knowledge infrastructure change means that the value of n or m changes.

Knowledge pattern change refers to a more moderate change. The input and output spaces remain untouched, but new patterns appear, or old patterns extinct, because of the addition of new data. An example would be change of handwriting styles of the same character. In figure 2.6 above, knowledge pattern change might be:

- The same input now produces a different output, for example, the specific instance in the figure now fall into category *Y3*
- New input patterns that are never seen before appear. For example, an instance with *X* values never seen before falls into category 2

Both types of changes are common in real world applications. By addressing these changes, the research of incremental data mining can greatly improve the adaptivity of a knowledge system to evolve with new data with minimum calculation cost.

2.3.2 Incremental Learning Research in Neural Networks

Adaptation of learning is a major focus of neural net research [Lipp 87]. As a result, the neural network research society is relatively more active in incremental data mining research. Early in 1989, McCloskey and Cohen described the "catastrophic interference problem" [McCloskey 89]. It was often addressed in later literatures [McClelland 94, Pessa 94]. In its simplest form, the problem may be stated as follows: when a network trained to solve task A is subsequently trained to solve task B, it "forgets" the solution to task A. In other words, in order to incrementally learn new patterns (here it is task B), the neural network will have to destroy previously built

knowledge structures. This problem was reported in many different applications including classification of remote-sensing data [Bruz 99] and handwriting recognition problems [Hebert 99].

From the perspective of incremental learning, the catastrophic interference problem is a *knowledge pattern change* problem. The reason that neural networks cannot deal with this problem is that the patterns the networks learn are distributed in every hidden layer nodes independently [Fahlman 90]. Therefore, it is impossible to change the network without affecting the previous learned patterns.

To overcome this problem, a conceivable solution is to specialize each hidden neuron by restricting them within localized regions of the input space. Radial basis functions (RFB) networks [Poggio 90] were derived from this idea. However, it usually requires a large number of neurons, and cannot deal with the infrastructure change since it cannot change the number of nodes. A combination of a feed-forward network and a self-organized map has been proposed as an alternative solution based on similar theoretical foundation [Hebert 99]. More works are being published in this area, and new generation of neural networks has stronger ability to deal with incremental changes than their precedents.

An interesting contrast to the active research in the field of *knowledge pattern change* is the almost "dormant" research field of *knowledge infrastructure change*. Although these changes are common in the real world, it seems that no effective solution has been found to deal with them yet.

In 1992, Ramani proposed an incremental learning method based on hidden layer activation mechanism [Ramani 92]. This method enabled a trained neural net to be reused for the construction of a new net that had more output nodes. Neural net built using this incremental approach showed similar performance compare to neural net built from scratch, but the training effort was reduced by a factor of 5. However, the problem addressed in Ramani's paper (fish classification) was a single outcome problem. Therefore, when samples belong to a new outcome class are added, the number of samples belonging to old outcome classes is not changed. We are interested to know how this algorithm performs in a multiple-outcome problem, where the addition of samples belonging to new outcome classes may also change the number of samples belonging to old outcome classes. More discussion about multiple outcome problems will be discussed in chapter 3.

2.3.3 Incremental Learning Research in Other Machine Learning algorithms

Neural network is not the only machine learning algorithm for classification tasks. Incremental learning has also been explored in combination with other classification algorithms. Interestingly, knowledge pattern change received more attention than knowledge infrastructure change in these research areas as well.

Syed proposed an incremental learning algorithm in Support Vector Machine (SVM) to handle concept shift [Syed 99]. By training the SVM using partitions of data set one at a time and keeping Support Vectors gained in each step, they were able to show that the algorithm can effectively learn the shifts in concept space, i.e., the knowledge pattern changes.

Decision tree is a classification technique that applies "divide and conquer" strategy to partition the problem space [Quin 88]. It has been applied to a range of applications including medical diagnosis, plant science and education evaluation. An incremental decision tree algorithm, ID5R, was proposed by Utgoff [Utgoff 89]. This algorithm could continuously learn from new data and adjust its knowledge structure based on them. ID5R is a case-by-case incremental learning algorithm, and is shown to have satisfactory performance for knowledge pattern change problems. However, no experiment for knowledge infrastructure change was performed using ID5R.

Researches on incremental learning are underway for many other types of machine learning algorithms such as Bayesian networks. However, since the focus in this dissertation is neural network, we will not cover them here. Incremental learning using other algorithms for IRES is currently under study in our lab.

2.3.4 Challenges of Incremental Learning

Based on the above literature review, we can see that few works have been done to deal with incremental learning associated with knowledge infrastructure change. In real world, such changes are ubiquitous: New categories of classification may be added. For example, a new genre of books may be added to a bookstore, new surgical procedures may be accepted by physicians. In addition, new input variables may appear. For example, certain patient demographic information may become available to the electronic patient record system. When the above scenarios appear, the learning ability of current approaches is seriously limited.

In this dissertation, we will develop an incremental neural network algorithm that addresses this problem.

2.4 Research Foundation

The theoretical foundations for incremental neural network (INN) algorithm developed in this paper stem from several properties of back propagation neural network. In this section, we will analyze these properties.

2.4.1 The Role of Hidden Layer in Back Propagation Neural Network

A back propagation neural network usually contains three layers of nodes. The input layer nodes present input values of an instance to the neural net, and the output layer nodes present the final result in the format of output categories that are activated or silenced. These two layers can be deemed as the "interface" that a neural net interact with the environment: They get the input from and present the output to the users.

The hidden layer, which resides in between the other two layers, is responsible for the intense calculations and the mathematical mapping from the input space to the output space. Two weight matrices are connected to the hidden layer nodes: the input-to-hidden layer matrix, and the hidden-to-output matrix. By adjusting these weight matrices, neural net tries find the function that is closest to the actual mapping function from input to output spaces.

Empirically, it was shown that after a neural network is trained, the input-to-hidden layer weight matrix captures the nature of the problem, and the hidden-to-output layer weight matrix "translate" this relationship back into output values.

The first theoretical assumption for this dissertation is that the hidden layer in a three-layer neural net acts as a feature detector. In a properly trained neural network, the weight matrix from the input layer to the hidden layer should represent a transfer function that captures the nature of the problem, not only the nature of the training sample used. The problem nature is reflected by the activations of the hidden layer nodes, which are calculated based on the input – hidden layer weight matrix. Therefore, in a three-layer neural network, the activations of the hidden layer represent the relationships between the input layer and the output layer [Ramani 92].

Consider the situation when a new output class is added. We argue that this relationship should still hold as long as the new class is not fundamentally different. In another word, the training procedure for the original neural network should capture the essential relationship between the input vector space and the output vector space, and the addition of this new output class should not change this relationship. Therefore, we can keep all the weights from input layer to hidden layer intact even if a new output class is added. Of course, there will be one additional output node in the output layer, and we still need to modify the weight matrix between the hidden layer and the output layer.

2.4.2 Back Propagation Neural Network's Ability to Learn Joint Outcome Tasks

Many real world applications, including IRES, have joint outcomes (also named as multiple outcomes). Decision instances in these applications have outcomes belonging to more than one decision outcome class. For example, a customer of a music store may be categorized as BOTH a jazz fan AND a rock music fan. A radiologist may categorize a certain image retrieval task as BOTH belonging to "most recent image" category and "same modality" category.

Traditional classification algorithms were developed to deal with single outcome tasks where each instance only belong to one outcome category. Many of them have limited performances when dealing with joint outcome classification tasks.

However, back propagation neural network can efficiently handle the joint outcome problems. As seen in figure 2.6, more than one outcome class (y1 to ym) may be activated simultaneously in one instance. In the next chapter, we will demonstrate that the ability to process multiple outcome instances is very important for the IRES classification task.

2.4.3 Summary of Research Foundation

Two conclusions can be drawn from the above analysis. First, the hidden layer's role as feature detector makes it possible to add in new outcome classes to a neural net without alternating its input-to-hidden layer weight matrix. The old knowledge can therefore be preserved while new knowledge can be gained through the training of the hidden-to-output layer matrix.

Second, back propagation network can efficiently deal with multiple outcome classification tasks. Since IRES is a multiple outcome classification application, this is one of the reasons that my dissertation builds on neural network for new algorithm development.

2.5 Research Questions

Based on literature review conducted in previous sections, I raise the following research questions to be answered in this thesis:

1. How to design and implement an incremental learning technique using back propagation neural network?
2. Can an incremental neural network model deal with multiple outcome classification tasks without losing performance?
3. If the incremental learning algorithm is implemented, how is its performance compared to traditional, non-incremental methods on the task?

In the next chapter, we will shift our attention from the algorithms to the specific application domain of Patient Image Retrieval for radiologists.

CHAPTER 3

AN OVERVIEW OF IMAGE RETRIEVAL EXPERT SYSTEM (IRES)

Retrieving a patient's prior examination images that are relevant to the current ones is a critical component in radiologists' primary reading services. It helps them to confirm their preliminary diagnosis, compare differential abnormal radiographic signs, and evaluate the progression of a known underlying pathological process or abnormality [Sheng 94a].

In the digital revolution of the radiology practice that is taking place all over the United States, such image retrieval support plays even more accentuated role. However, most of the current image retrieval practices are inadequate.

Researchers and clinicians alike have extensively investigated many approaches to help improve prior image retrievals' efficiency. Among them there are statistics-based approaches and knowledge-based approaches. While mathematically sound, the statistics-based approaches often lack desirable medical/radiological foundation [Kishore 92]. Knowledge-based approaches gain solid medical foundation through knowledge acquisition process with domain experts. However, they experience a range of challenges including radiologists' difficulty to express underlining knowledge, wide variations in individual radiologists' retrieval preference, and the need for easy and frequent modification of knowledge (incremental learning). Recently, data mining techniques

were explored as an alternative in order to overcome the above problems faced in traditional knowledge acquisition techniques [Sheng 94a & 98].

In this chapter, we will briefly review the primary radiology image reading process, present an overview of IRES, and analyze problems that exist in current data mining techniques in IRES.

The next section explores problems experienced in current film-based image reading process and explains the importance of IRES in digital radiology. Section 3.2 examines characteristics of image retrieval knowledge. This will be followed by an overview of previous IRES research works. Section 3.4 discusses the need and opportunity for incremental data mining in IRES.

3. 1 The Image Reading Process and Prior Image Retrieval

Radiologists provide physicians and other specialists with timely reports on findings from their examinations and radiology procedures, together with interpretations and patient management recommendations [Sheng 98]. Recent advances in medical imaging technology have made radiology examination increasingly informative, descriptive, and affordable, and they are becoming central to effective patient care and management.

A cognitive model of a radiologist's image-reading process was built based on a verbal protocol analysis and substantiating interviews and is supported by observations of actual readings [Sheng 94c]. According to this model, radiologists' reading process can

be broken down into four steps: examination orientation, hypothesis generation, hypothesis validation, and report compilation.

During the *examination orientation*, the radiologists read a consultation requisition form and an examination form that tell them about the referring physicians' clinical questions and the patient's clinical history as well as status. Based on this information, the radiologists will *generate hypothesis* about the underlying disease or abnormality. They then attempt to *validate the hypothesis* by comparing current images with relevant prior ones and reviewing additional patient information. This process can be repeated until they are confident with their diagnosis. They then *compile a report* that interprets the image, summarizes their findings, and provides patient-management recommendations. Prior images are needed to validate the hypothesis during this process.

Two types of techniques support retrieval of prior images: the statistical-based approach derives image-retrieval statistics from image reference logs that record past image retrievals. Most statistical solutions are fairly unsophisticated and thus can not meet the diverse reading needs.

Another technique, the heuristics-based approach, assumes that the image retrieval decisions are based on some set of heuristics. Thus, identifying these heuristics and presenting them as rules should satisfy the radiologists' needs. Production rules [Levin 90], decision tables [Kishore 92] are some of the examples of heuristic approach.

The time and difficulty to initially identify the knowledge and maintain the knowledge is a major problem for heuristic based learning. Data mining techniques can reduce the time required to engineer knowledge by many orders of magnitude. In the next

section, we will analyze the characteristics of the image retrieval process and introduce suitable data mining techniques to support IRES.

3. 2 Characteristics of Image Retrieval

Developing a learning solution for a particular task requires determining the type of data mining problems the task belongs to, characteristics of the target application, and input and output attributes. In IRES project, the first step is to determine what type of problem image retrieval belongs to.

3. 2. 1 Mapping of IRES Problem to Classification Task

Based on Figure 2.3 we can determine what type of data mining task image retrieval needs. First, the goal of the image retrieval activity is to *predict* prior images that are needed based on past experience, not to describe or categorize images or patients. This leads us down to the left branch of the tree in Figure 2.3.

The next question is: "Map to a real value or predefined category?" According to the knowledge acquired from previous study, the ultimate goal of IRES is to map the output to predefined image retrieval category, for example, "the most recent, same modality, same anatomical portion image". Therefore we come to the conclusion that the IRES system needs classification data mining techniques (categorization) to help.

As a classification problem, IRES demand clear definition of the input attributes as well as the decision outcome classes. Based on previous study [Sheng 94b & 94c], the following characteristics are derived:

- *Input Attributes:* We adopted a reason – for – examine approach to understand the image retrieval behaviors [Sheng 98]. Based on this approach, the knowledge can be derived from the information contained in the examination forms and consultation requisition forms. Table 3.1 summarizes the input attributes of learning related to image retrieval.

Category	Input Attributes
Current Examination Related	• Examined anatomical portion of the current examination • Modality of the current examination (e.g., X-ray, CT, etc.) • Reason for examination (e.g., disease diagnosing, screening, etc.) • Availability of the most recent exam with the same modality and anatomical portion as the current examination
Patient Related	• Gender • General Condition (e.g., satisfactory, critical, etc.) • Clinical Status (e.g., urgent, short term, or long term) • Patient source (e.g., ER, etc.) • Pregnancy Condition • Use of alias (for trauma patients)
Disease/ Abnormality Related	• Disease/Abnormality Type (e.g., mass, vascular, stone, etc.) • Disease Nature (e.g., congenital or acquired) • Disease Phase (e.g., acute, sub-acute or chronic) • Mass type, if the disease type is mass (e.g., malignant or benign)

Table 3.1 Input Attributes for Patient Image Retrieval Knowledge Learning

- *Decision outcome classes:* As Table 3.2 shows, radiologists' decision outcomes can be classified into three dimensions. Because these dimensions can jointly describe a radiologist's image retrieval, they will be the basis for the decision outcome classes. Since retrieval of more than one image is common practice, multiple outcomes should be expected. A total of 26 outcome classes seen in common practice are chosen, each represents a specific combination of the three dimensions below.

Dimension of Prior Examination	Possible relationship with current examination
Anatomical Portion	-- Same Anatomical Portion -- Related Anatomical Portion
Modality	-- Same Modality -- Related Modality
Time/Sequence • Recency	-- One most recent prior examination -- Two most recent prior examinations -- Three most recent prior examinations
• Specificity	-- Baseline examinations -- Most Recent Pre-operation examination
• Time Interval	-- One Week Prior -- Two Weeks Prior -- Three Months Prior -- Six Months Prior -- One Year Prior

Table 3. 2 IRES Decision Outcome Classes

3. 2. 2 Characteristics of IRES Classification Task

Patient image retrieval poses several challenges as a classification learning system:

- *Incomplete and noisy input information:* Some input attributes may not apply to certain examinations or patients. In some cases, some attribute cannot be derived directly from the available forms.
- *Multiple Decision Outcomes:* Radiologists often reference patient images from more than one prior examination. When following up on patients with a particular benign brain tumor, for example, they may want to reference images taken three months apart, dating back from the baseline examination.
- *Inconsistent Image-Retrieval Outcomes:* Specific image retrieval behaviors may vary across radiologists. In another word, the image retrieval behavior is highly individualized. Besides, given the fact that radiologists' experience increase over time, the image retrieval behavior is also dynamically evolving. As a result, even for the same person and same input information, the outcome decision might change over time.

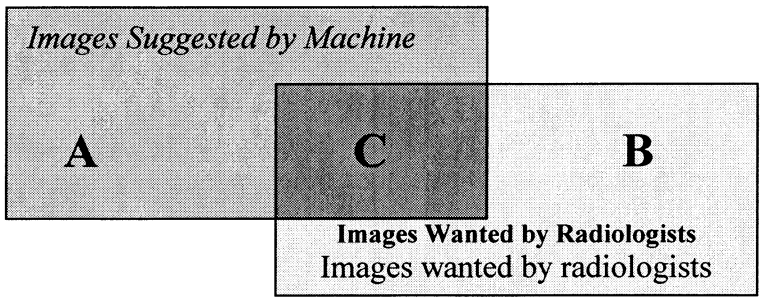
Considering the above characteristics of IRES classification, two classes of algorithms were chosen in our previous research: neural network, for its tolerance with missing inputs and its capability of handling multiple outputs, and decision tree, for its fast learning ability and expressive explanation of reasoning.

3. 3 Overview of Previous Works in IRES Project

IRES project started around 1990. Many researches including knowledge engineering, expert system building and machine learning were carried out. In this section, we focus on the machine learning algorithms designed for IRES classification task and their performances.

3. 3. 1 Performance Evaluation Metrics

Performances of classification algorithms are usually measured by their ability to correctly classify new instances. In the specific case of IRES, the accuracy is multi-faceted. The power and efficiency are two important dimensions. They can be measured by recall rate and precision rate respectively. The recall rate is the percentage of the images referenced by a reading radiologist that the learning system correctly suggests. The precision rate is the percentage of images the learning system suggests that are actually referenced by a radiologist. Thus, the recall rate demonstrate the power of the learning system by revealing its false negatives, while the precision rate demonstrate the efficiency of a learning system and is closely related to its level of false positives. Figure 3. 1 depicts the relationships between precision and recall rate.



Recall rate = Area C / Area B **Precision Rate = Area C/Area A**

Figure 3. 1 Recall rate and Precision rate

It is plausible that there is a trade-off between recall rate and precision rate. For example, the precision rate can be increased by increasing selective retrievals. However, this will often lead to a lower recall rate. Similarly, the recall rate can be increased by retrieving more images, with the side effect of lowering the precision rate.

3. 3. 2 Performances of Previous Machine Learning Algorithms

We conducted experiments in UMC involving 200 cases whose clinical readings had been performed by radiologists or residents. Each case contains sufficient information to code input attributes and document patient image retrieval behaviors. Using this data set as training and testing data, the performance of a neural network and a multiple decision tree algorithms is measured.

Since each output node in the neural network will produce a signal between 0 and 1, we need to change it to either 0 or 1, standing for "not retrieve" and "retrieve" respectively. Therefore, the variable *retrieval threshold* is introduced. Any value that is greater than this threshold will be deemed as 1, while any value smaller than it will be deemed as 0.

Retrieval Threshold	Recall (%)	Precision (%)
0.5	72.05	74.25
0.4	74.30	69.76
0.3	75.47	68.26
0.2	78.84	64.87
0.1	80.50	58.49

Table 3. 3 Learning Performance of the neural network

As shown in Table 3.3, the neural network's performance is comparable to that of the radiologists'. We can also see that with the increase of threshold, recall rate gets lower, while precision rate is improved.

The training of multiple decision trees resulted in similar performance as shown in Table 3. 4. However, the tree approach outperformed the neural network in both the recall rate and the precision rate. They are both capable of learning patient image retrieval knowledge. Furthermore, their performance is expected to increase with increments in the size of training samples.

Critical Value	Recall (%)	Precision (%)
0.5	81.86	63.94
0.6	76.47	69.46
0.7	75.03	73.93
0.8	73.46	74.88
0.9	71.80	75.36

Table 3. 4 Learning Performance of the Multiple Decision Tree Approach

So far, we have examined the image retrieval behavior by knowledge engineering approaches and built a knowledge base that had comparable performance to the radiologists' retrieval activity. We have also used two data mining techniques to extract knowledge from real world data. They both have comparable performances to that of the knowledge-based approach. They also have higher flexibility and require less time and effort to maintain.

The data mining techniques we used in previous studies are not without their weaknesses. One vital problem for these techniques is that they cannot deal with dynamically changing patterns efficiently. The next section will be devoted to a discussion of this problem.

3. 4 Current Problems in IRES

The ultimate goal of IRES project is to build an adaptive and active online decision support system. An active system is a system that can adapt to the dynamically evolving patterns. An adaptive system is a system that can respond to individual users according to their own styles and behaviors. Such a system will inevitably encounter the incremental learning problem. Therefore, new generation of data mining algorithms that can learn incrementally is needed.

During the interview with the radiologists, we often heard comments like the following:

-- "I would consider this factor if I had access to this information."

-- "If possible, I want the system to tell me more outcome decisions than it can do now (For example, tell me which image to retrieve instead of which exam)."

These comments imply the potential changes that might occur in the IRES data. These changes fall into the category of knowledge infrastructure change: The input or output space will be changed. The first comment represents the scenario in which new information is available and the input space need to be enlarged, while the second comment describes a situation where the addition of output classes is required.

As reviewed in chapter 2, not many algorithms can perform the type of incremental learning required by IRES. Specifically, when new data with changed knowledge infrastructure come in, the neural network developed before has to be thrown away, and a new neural network has to be built based on the new data set and the old data set. Since construction and training of a neural network is very time consuming, this is not a feasible approach to solve the problem. In the next chapter, we will develop a new incremental neural network to overcome this obstacle.

CHAPTER 4

DEVELOPMENT OF INCREMENTAL DATA MINING TECHNIQUE: THE INCREMENTAL NEURAL NETWORK APPROACH

The neural network algorithm previously developed for IRES is limited to domains where the input and output spaces are static, and the addition of new data does not change the knowledge infrastructure. When the system environment or the characteristics of the data change, it cannot adapt. The only solution would be to throw the trained network away and build a new one from scratch based on the new environment. Since the training of neural network is extremely time consuming, such a solution is very costly. In this chapter, we will develop an incremental neural network that utilizes the trained neural network to save learning time without reducing learning performance.

4. 1 Knowledge Infrastructure Changes in the IRES System

4.1.1 Changes That IRES System May Encounter

The IRES system might encounter both knowledge infrastructure changes and knowledge pattern changes. An example of pattern change is radiologists' evolutionary retrieval behavior: Over time, same radiologist may have different outcome decisions

made based on the same input information, simply because of increased experience. Another example would be the introduction of new image retrieval practice patterns that is never encountered by the IRES system before.

There may also be knowledge infrastructure changes in IRES as well. More information will be available in the hospital information systems in the near future. This information may be desirable for the IRES system but was not available before. To add this information into IRES requires changes to be made in the input layer structure. Also, new outcome decision classes may appear due to more detailed knowledge engineering research or addition of new equipments in the future. This can result in a change of the output layer structure.

To build an active and adaptive IRES system, we have to have algorithms that can deal with the incremental learning problems mentioned above. This dissertation represents part of the research efforts towards an incremental learning IRES system in our lab. Specifically, it deals with the change of output layer structure.

4.1.2 Change of the Outcome Decision Classes

We introduced the 26 outcome classes that are extracted from the IRES knowledge base in Table 3. 2. All past algorithms were trained using these classes as potential outcomes. These 26 outcome classes were all medically meaningful combinations of the three decision dimensions introduced in Section 3.2.1 at the time of the knowledge acquisition process, not a list of all the possible decisions. Consider the situation when a specific combination of the three dimensions that was not medically

meaningful before just became meaningful due to addition of new equipment or introduction of new retrieval knowledge into practice. Now we will have 27 outcome decision classes. The standard solution is to discard the existing network and simply train a new network with 27 outcome classes. The previous training effort is wasted. When the IRES system goes online, the extra time needed for the training of such a new network could greatly slow down the whole system.

It is worthwhile to mention that the above scenario is not just a hypothetical situation for the pure fun of research. Our interview with the radiologists showed the potential new outcome classes that might be of interest to the radiologists. One example is that they wanted to know which *image* to retrieve rather than which *examination* to retrieve.

There is only one paper that specifically addressed this type of data mining problem [Ramani 92]. We will adapt the algorithm developed in this paper to solve the problem stated above. It is worthwhile to notice, however, that the algorithm developed in Ramani's paper only deals with single outcome tasks. The challenge brought by multiple outcome nature of IRES tasks will be discussed in the next section.

4.2 The Incremental Neural Network (INN) Algorithm

4.2.1 Hidden Layer Activation

The primary purpose of the hidden layer in the neural network is to act as a feature detector. Thus, in a properly trained neural network, the transfer function

represented by the hidden layer weights should depend on the structure of the problem, not just the training sample used [Ramani 92].

If we postulate that this relationship should still hold even if a new category is added to the classification problem, the hidden layer weights should still be able to represent the nature of the problem. Therefore, rather than discarding all of the hidden layer weights already trained, we could reuse them.

The procedure of hidden layer activation learning is as follows: First, combining all the data including the new data with additional outcome classes together. For the old data, simply add 0's to the extra output classes. Second, use this data set to compute the activation of the hidden layer of the old neural network, i.e., use the input values from the data set to calculate the values of the hidden nodes in the old neural network. Third, use the activation values as inputs, the output values from the data set as actual outputs, train a two layer neural network. Finally, simply append the new two layer neural network to the bottom half of the old neural network to form the new neural network. Figure 4. 1 describes the algorithm.

4. 2. 2 Previous Work in Incremental Neural Network Algorithm

In his paper, Ramani used the INN algorithm described above to solve a fish identification problem. The data of 6 categories of fish was used to train a three layer neural network for fish identification. The resulted network was able to reach an accuracy of 95.8 percent. Then a hypothetical 7^{th} fish category was added to the data set. Using the INN approach, Ramani was able to train a neural net that had about the same

performance as the traditional, built-from-scratch one. The training effort of incremental learning measured by training time is by a factor of five.

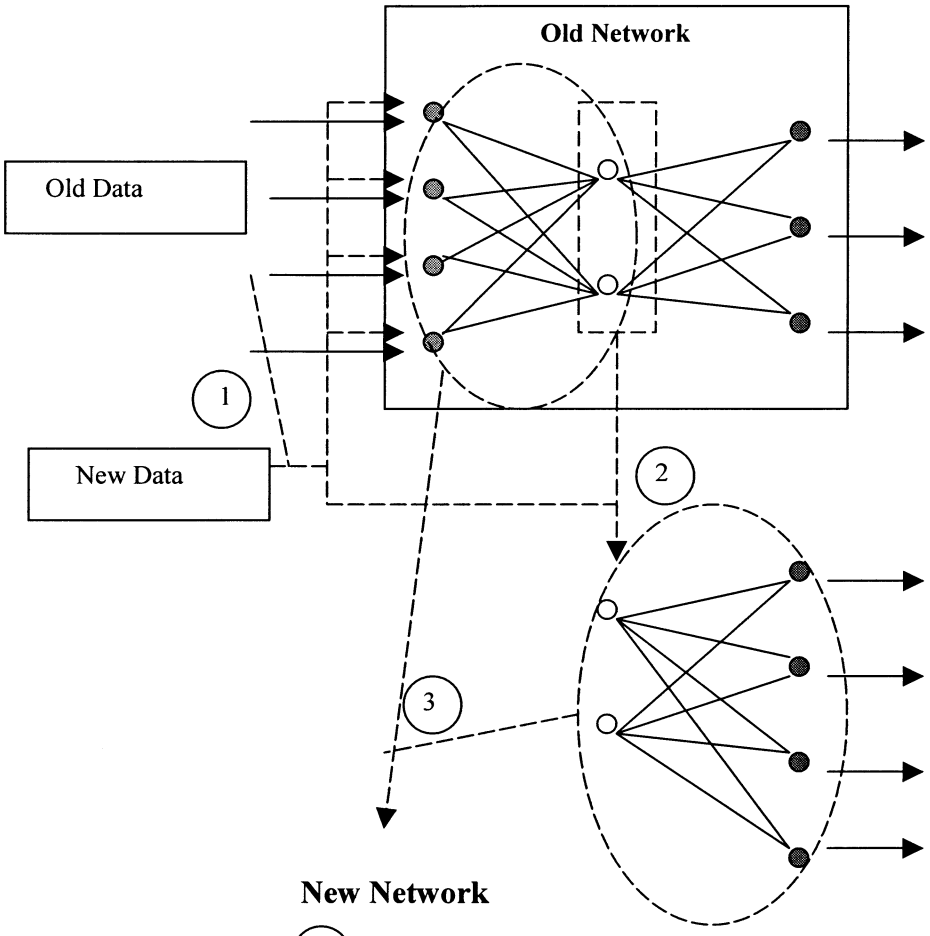


Figure 5. 1 The Incremental Neural Network Learning Process

The significant reduction in training time achieved in Ramani's paper is the major motivation for the use of INN in IRES system. However, there are many challenges to the INN algorithm in the IRES application domain that were not studied by Ramani:

- Ramani's research used same number of examples for each output category during the training. That is, for the six categories of fish, each category has same number of examples. The seventh category also has the same number of examples. This might be due to statistical concern, but the real world data set is not like this. For example, IRES data does not have such a statistical characteristic. It is of our interest, therefore, to observe the performance of the INN algorithm in a real world setting, where the additional class may have far more, or far less examples available than other classes.
- Ramani's network only dealt with a single output problem. His goal was to classify fish into one of six or seven categories. In the IRES system, we deal with multiple outcomes. More than one outcome class may be chosen in one example. This greatly complicated the problem. However, if the prerequisite that the hidden layer weights capture the problem nature holds, INN should be able to deal with multiple outcome problems. The ability of INN to handle multiple outcome problems is another research interest of ours.

Although single outcome vs. multiple outcomes seems to be a trivial difference for the application, it has profound impact on the training of the incremental neural

networks. In a single outcome problem, if new data containing a new outcome class are added, they will NOT affect the other outcome class patterns. In another word, assume that the input-hidden layer matrix captures the patterns for the original outcome classes, it would not receive any new instances for these classes in the new data set. Therefore, it is quite safe to say that this matrix can be applied just the same to the new data set when old outcome classes are involved. The trained hidden-output layer network, in this case, will capture the information of the additional new outcome class pattern.

However, such is not the case for multiple outcome tasks. Consider a simple scenario where we have two outcome classes to start with. With two classes there are four possible classifications: 10, 11, 01 and 00. Assume that the network is trained to capture these patterns in its weight matrices.

Now, a third class is added. This brings in many other classifications: 100, 110, 101, 111, 010, 001, 011, and 000. The critical observation is that in the new training cases, which contain "positive" for the new class (111, 011, 001 and 101), old classes may also appear. This means additional patterns for the OLD classes are also added when the new data run through the network. An important question is: will this affect the patterns for the old classes? As a result, will the input-hidden layer matrix still be an effective feature detector for the new data set?

In the next chapter, we will use IRES data to empirically answer the above questions.

CHAPTER 5

PERFORMANCE EVALUATION OF INN ALGORITHM

With the theoretical foundation laid out and algorithm design introduced, we now start to implement the INN algorithm. This chapter is organized as follows: First, we will discuss the process of building a neural network with satisfactory performance for the IRES application. This serves as foundation for all experiments carried out in later sections. Then we will go through a series of experiments to evaluate the INN algorithm.

5. 1 Training and Parameter Tuning of Back Propagation Neural Network

To work with either incremental neural network or a network built from scratch both requires tuning of the network parameters to reach good performance at the beginning. In this section, we walk through experiments that are carried out to find the best architecture for a back propagation neural network for IRES application.

5.1.1 Data Set and Training Procedure

The data set used in these experiments is the same 200 cases used in the previous study for IRES machine learning. A benefit is that we can benchmark the performance our neural network using that of the "old" neural net developed in previous study.

The network was trained using a set of training cases and tested using separate testing cases. Because a single train-and-test experiment may generate misleading performance estimates when the sample size is relatively small, we used a ten-fold cross-validation technique [Weiss 91]. The 200 cases were randomly divided into 10 exclusive data groups of equal size. In each train-and-test process, we chose one data group as the testing group, and the remaining nine groups became the training set. We then performed the training ten times and estimate the performance of the net by averaging performance results from the 10 training processes.

5.1.2 Experiment Results

Experiment 1. Preliminary Comparison with the Previous Network

Goal: To establish some preliminary estimates of the performance of the back propagation neural network

Procedure: All parameters were set to the values reported in the previous study. The number of hidden nodes is 100, the learning rate started at 0.25, and the momentum is 0. Threshold was set to 0.5. The result is shown in table 5. 1.

Neural Network	Avg. Precision Rate	Avg. Recall Rate
Previously Designed	74.25	72.05
OO designed	74.85	71.35

Table 5. 1 A comparison of Performance between Previously Designed and New Neural Network

Result. From this simple test we knew that the performance of the new neural network is comparable to that of the previously designed one. Therefore, we went on to tune the parameter of the new neural network.

Experiment 2. Find Best Hidden Node Number

Goal: Find the optimal hidden node number for IRES application.

Procedure: Using a learning rate of 0.25, momentum of 0 and threshold of 0.5, different hidden node numbers were used to construct the neural network. We wanted to find the hidden node number that has the best precision rate and recall rate.

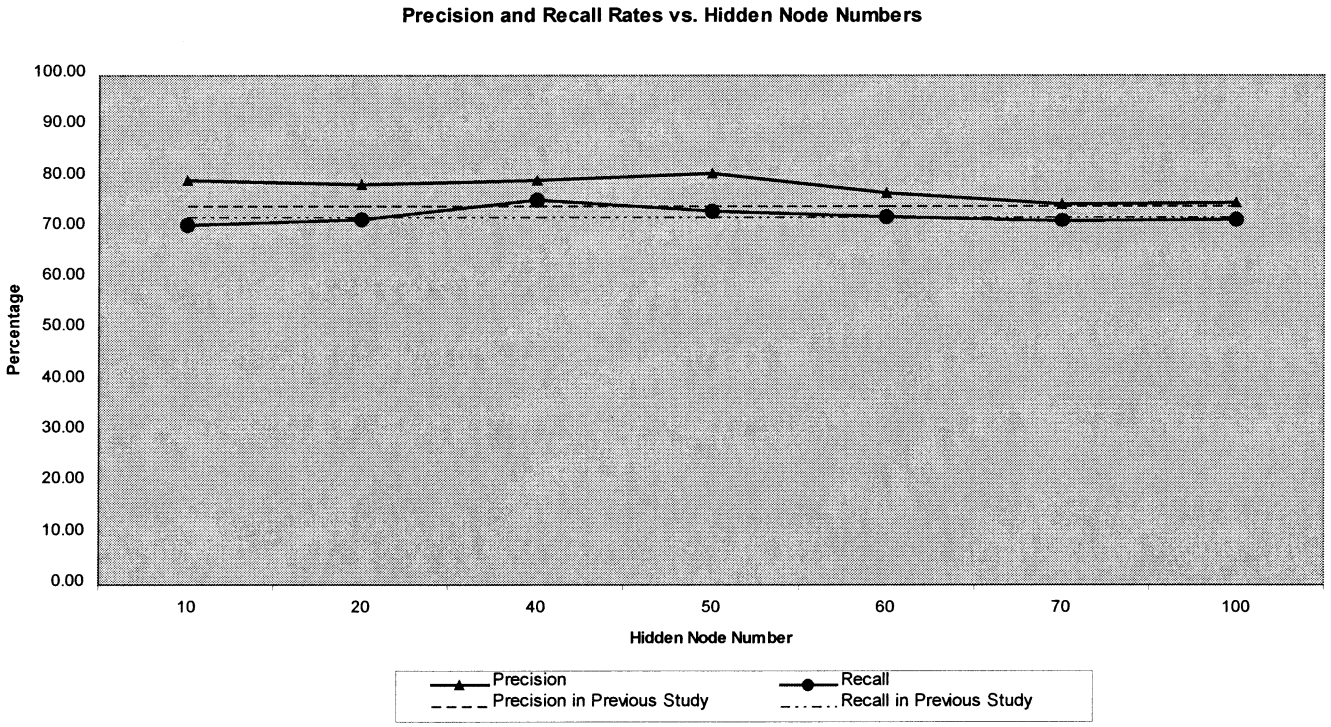


Figure 5.1 Precision and Recall rates vs. Hidden Node Numbers (Learning Rate = 0.25)

Results: As shown in Figure 5.1, the highest recall rate and precision rate were achieved using a network with about 40 to 50 hidden nodes. It is also shown in the figure that when using 100 hidden nodes, our neural network performs as well as the previous neural network. The previous student chose to use 100 hidden nodes as optimal setting for his experiments. Given the fact shown above, we decided to use a hidden nodes number around 50 as optimal setting. Experiment 4 was designed to determine the exact number of hidden nodes we need.

Experiment 3. Find the Optimal Learning Rate

Goal: Find the learning rate that leads to better recall and precision rate.

Procedure: Since we have not decided on the number of hidden nodes yet, we chose two settings, 50 and 100 hidden nodes, to train the network with learning rate of 0.25, 0.50, and 0.75. The result is shown in Figure 5. 2.

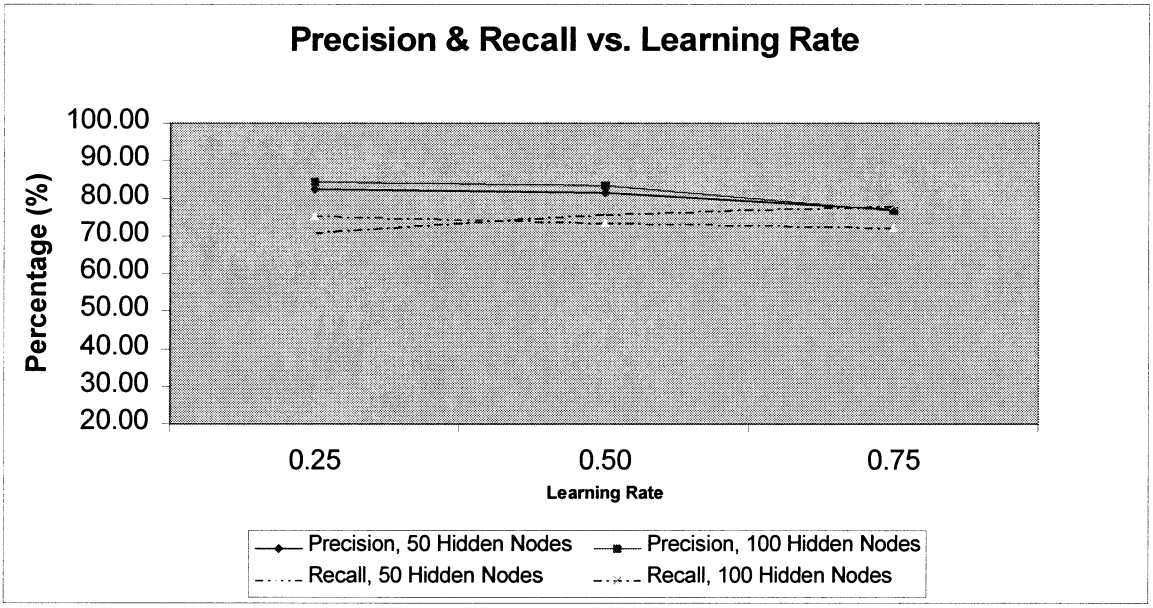


Figure 5.2 Precision and Recall vs. Learning Rate (hidden nodes number = 50 and 100)

Result: When using 100 nodes in the hidden layer, the precision and recall rate decrease as the learning rate increase. This decrease is especially significant when the learning rate changed from 0.50 to 0.75. When the hidden layer contains only 50 nodes, the changes in precision and recall vs. the change of learning rate is more complicated. While the recall rate increases with the increase of learning rate, the precision actually decreases. To balance these two different changes, a learning rate of 0.5 seems to be best when hidden nodes number is around 50.

Experiment 4. Fine Tuning of Hidden Nodes Number

Goal: To find the exact number of hidden nodes that is optimal

Procedure: A learning rate of 0.5 is used because of the results shown in experiment 3. 10, 40, 45, 50, 60, 70 and 100 hidden nodes were used to construct the network.

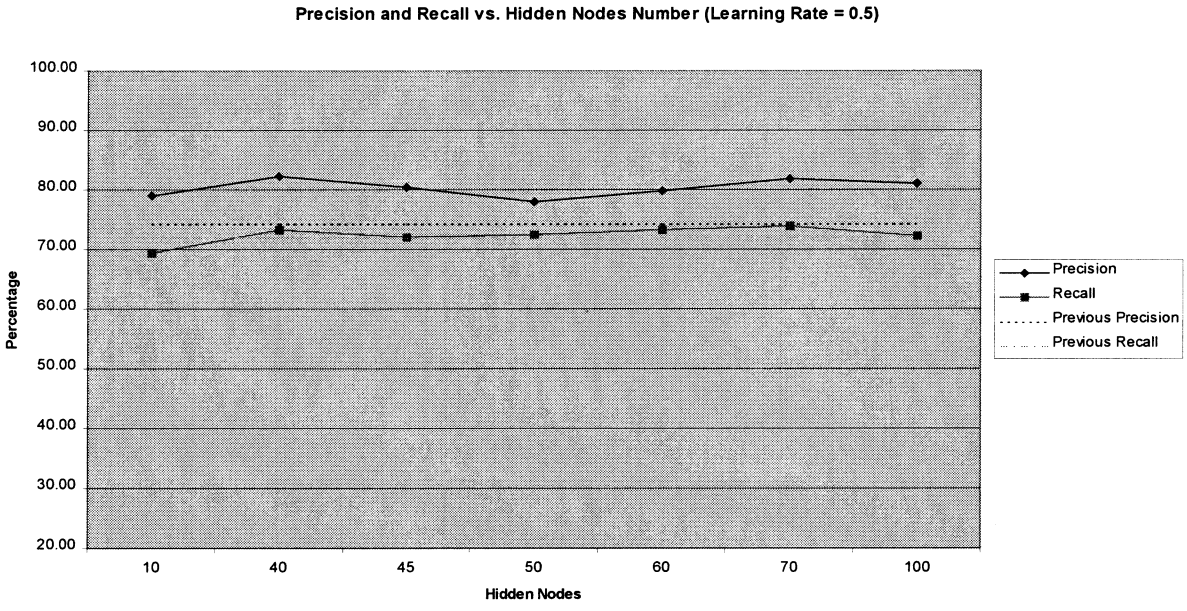


Figure 5. 3 Precision and Recall rates vs. Hidden Node Numbers (Learning Rate = 0.5)

Result: Both 40 and 70 hidden nodes network produced very good precision and recall rate. The precision and recall rates of neural networks containing other numbers of hidden nodes were also comparable to those of the previously implemented neural network. An empirical method to determine the hidden node number is:

Hidden Nodes Number = (Input nodes number + output nodes number) / 2

In the IRES system, there are 64 input nodes, and 26 output nodes. Therefore the optimal hidden layer nodes number should be around 45. This experiment supported this empirical rule.

As a result of the experiments performed above, we decided to use the following parameters for the future experiments, including the incremental neural network experiments: Learning rate is 0.5, momentum is 0, hidden layer nodes number is 45.

5. 2 Performance Evaluation of INN

5.2.1 Experimental Design

- Data Set: To ensure that the data comes from real world, we used the 200 diagnosis cases gained from UMC and used in previous studies as well as in last section. To simulate the incremental learning problem, we got rid of all examples in the data set that contain one certain outcome decision class. Now we have a data set with 25 outcome decision classes. We trained a neural network with such

a data set, and then added back the examples of the 26^{th} class. This whole data set will then be used to train a traditional neural network built from scratch and an INN network. The performance of these two networks will then be compared.

- Evaluation Criteria: Precision rate and recall rate will still be used as evaluation criteria. Since we want to know whether the size of the new category has an effect on the performance of the INN algorithm, we also included a variable named "incremental ratio" to measure the increase in the amount of information:

  Incremental Ratio = size of the new category / size of the old data set

  For example, if we have a data set containing 150 examples, and the new category has 50 examples, then the incremental ratio will be 33.33%. In a single output problem, this ratio can accurately reflect the potential effect of the new category on the old data set. In a multiple output problem, however, this ratio can only approximate the new category's effect, because the new examples may also have some other old categories in the output.

  To measure the training effort, we defined a "time reduction factor":

  Time Reduction Factor = training time of the traditional net/ training time of the INN net

  The bigger this factor is, the more improvement we achieved by using the INN algorithm.

- Training Scheme: All of the neural net in the experiments will use the following parameter settings: learning rate = 0.5, momentum = 0, number of hidden layer nodes = 45, threshold = 0.5. Please refer to experiments done in previous section for the reason to choose this scheme.

In each of the following experiments, data containing one of the 26 outcome classes was removed from the data set. The remaining data set was then split into four equal size groups. Three of them would be used as training data while the one that remains would be used as testing data. Four training-and-testing cycles would be carried out, and the performance of the network was evaluated. If the performance was comparable to that of the previous one, then we conclude a neural network for 25 output classes was trained successfully.

Once the neural net for 25 output classes was trained, an incremental neural net will take the stored structure of the 25-output net, and run through the 200 cases (with the new category added back now) using INN algorithm. The training for the INN follows the same scheme as the training for a traditional neural net: A five-cross validation scheme similar to the ten- cross validation described in Chapter 4 will be performed. In another word, the 200 cases will be divided into 5 groups, four of them for training and one for testing. The result of this INN training will then be compared with the result of a traditional neural net training using the 200 cases and same training schemes.

5. 2. 2 Experiment Results

6 groups of experiments were performed. Each experiment got rid of examples belonging to one outcome decision class. The first outcome class, "retrieval of the most recent one image in the same modality in the same anatomical portion", has more than 150 examples. Its removal will only leave less than 50 examples for the training of the 25-output-class net. Therefore, the removal of this output class was not tested. Outcome class 9, 10, 14, 15, 19 and 20 don't appear in the dataset at all. Therefore, their existence does not have any effect on the training. The removal of these output classes was not necessary.

Figure 5.4 shows the distribution of output classes in the 200-example data set. There are five different types of outcome decision class judged by their incremental ratios: Class 1 is unique in that its incremental ratio is bigger than 1. Class 2 and 6 belongs to "strong incremental" category, and their incremental ratios are both greater than 50%. Class 7 and 11 both have incremental ratios around 20%, and can be called "moderate incremental". Class 3 and 16 incremented by about 10%, we rank them as "weak incremental". The rest of the classes have very small incremental ratios such as 2% or 4%. The removal or addition of such classes should not cause a qualitative change in the problem space. Therefore, they are not really incremental problems from this perspective. Table 5.2 shows the classification of outcome classes.

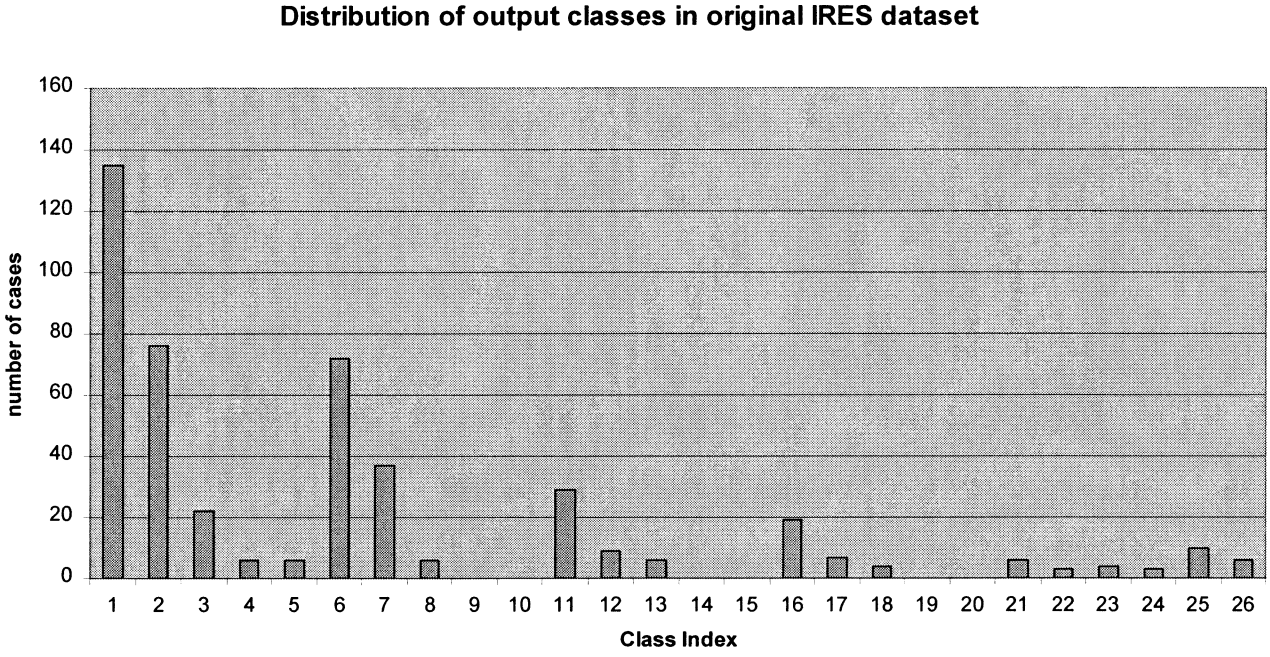


Figure 5.4 Distribution of output classes in original IRES dataset

Output Classes Category	*Incremental Ratio*	*Classes*
Very Strong Incremental	> 100%	1
Strong Incremental	> 50%	2, 6
Moderate Incremental	~ 20%	7, 11
Weak Incremental	~ 10%	3, 16
Not Incremental	< 5%	4, 5, 8, 9, 10, 12 ~15, 17 ~ 26

Table 5.2 Categories of Outcome Classes in IRES Data Set

Results:

A. The performance of the 25-outcome-class net:

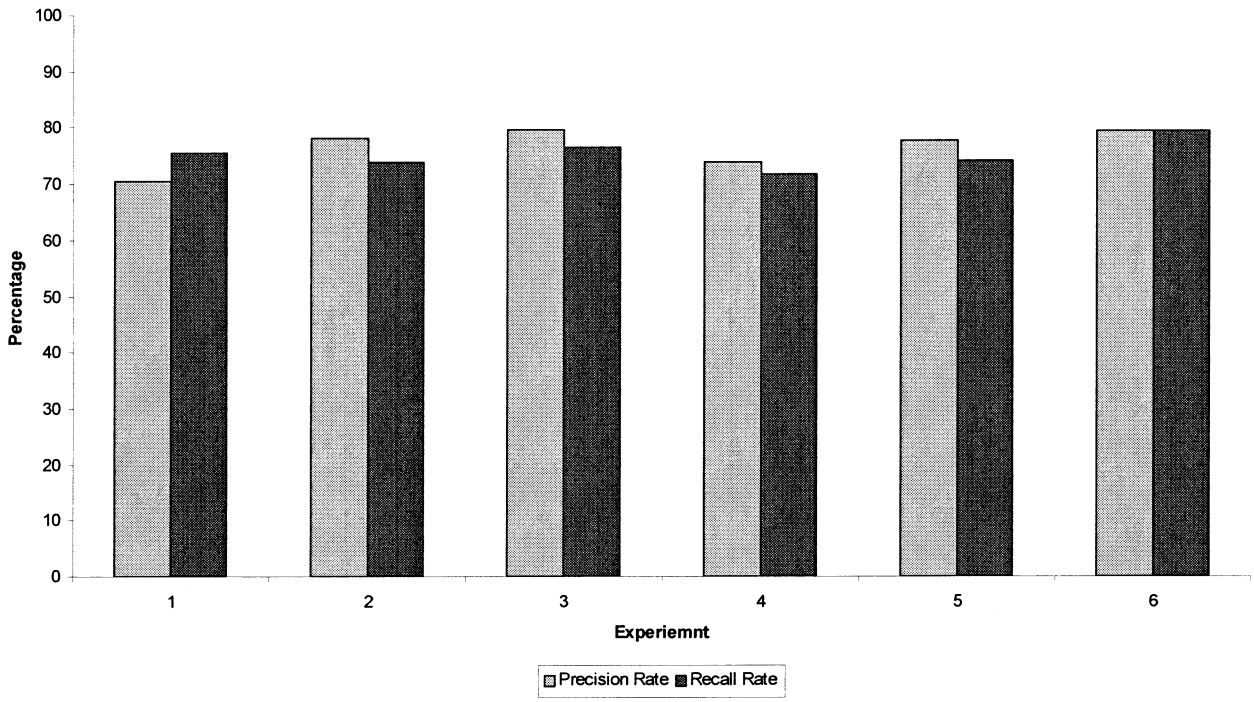
The first thing we need to measure is the performance of the 25-outcome-class net. Since the hypothesis that this net can capture the nature of IRES problem is the prerequisite for the INN algorithm, the performance of it is critical for the success of the INN algorithm.

Table 5.3 and Figure 5.6 show the performance of the 25-outcome-class nets built in the six experiments. Although the outcome space has changed, the performance of these nets should still be comparable to that of the 26-output-class net to demonstrate that they captured the problem nature. Therefore, the precision rate and recall rate of the 26-outcome-class neural net built in chapter 4 was used as control.

Experiment	Precision	Recall
1	70.50	75.52
2	78.08	73.95
3	79.54	76.40
4	73.85	71.68
5	77.81	74.02
6	79.44	79.44

The precision of 26-outcome-class net: 80.32
The recall of 26-outcome-class net: 71.94

Table 5.3 The Performance of 25-outcome-class nets



The precision of 26-outcome-class net: 80.32

The recall of 26-outcome-class net: 71.94

Figure 5.6 The Performance of 25-outcome-class nets

All the 25-outcome-class nets have lower precision rates than the 26-outcome-class one. But the performances were comparable because the differences were not significant. The recall rates of the 25-outcome-class nets were comparable to that of the 26-outcome-class one. Therefore, we conclude that these nets have captured the nature of the problem and were qualified to serve as the foundation for the INN learning. This also demonstrated again the power of neural network technique in the IRES problem domain.

B. The Performance of INN nets

Figure 5.7 and Table 5.4 summarize the performance of the INN nets constructed in the six experiments measured by precision rate and recall rate.

Experiment	Class	Precision	Recall
1	2	76.73	76.58
2	3	75.43	75.35
3	6	70.79	66.64
4	7	79.32	75.12
5	11	76.80	72.18
6	16	77.58	73.32

Table 5.4 Performance of INN Algorithm in 6 experiment groups

It is evident that the performances of the INN nets are comparable to that of a net built from scratch on both performance measures. Again, we observed the fact that the precision rates of INN nets were all lower than that of a traditional one. This limitation may be due to the lower precision rate of the 25-outcome-class nets that served as the building blocks for the INN nets. We then compared the performances of the INN nets to those of the 25-outcome-class nets.

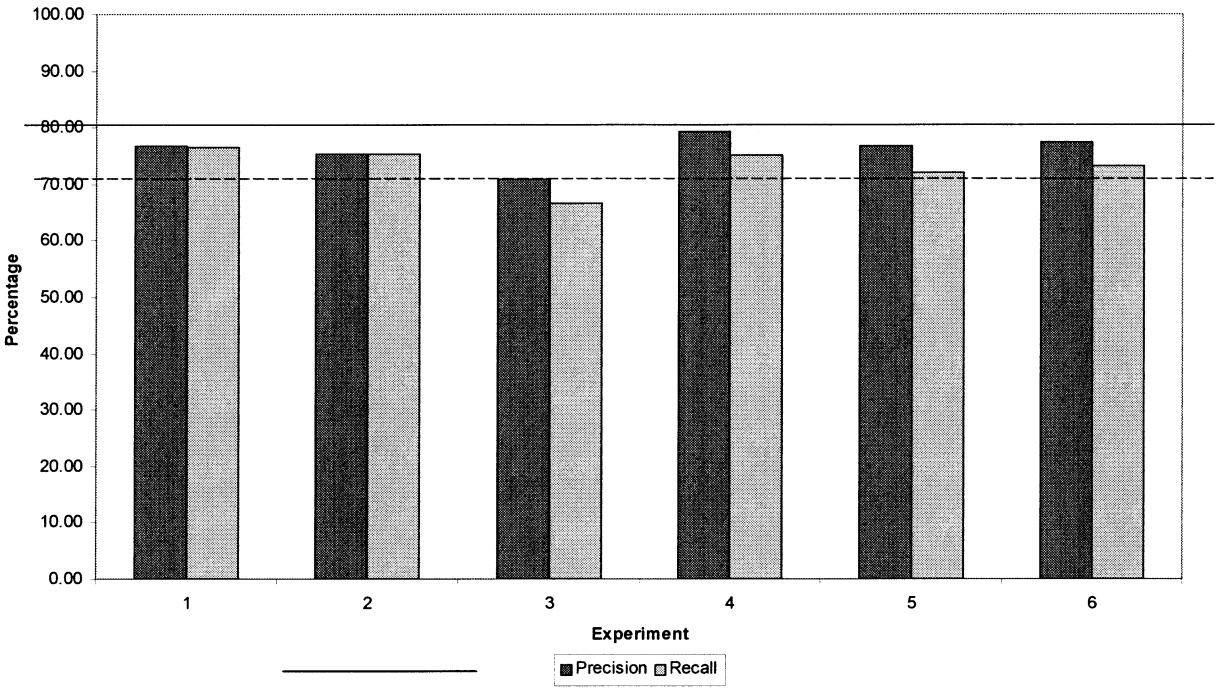


Figure 5.7 Performance of INN Algorithm in 6 experiment groups

C. Time Complexity.

Category	Class	Incremental Ratio	Time Reduction Ratio
Strong Incremental	2	58.73	4.9101
	6	53.85	4.2256
Moderate Incremental	7	21.21	4.6789
	11	15.03	4.4442
Weak Incremental	3	11.11	4.2459
	16	9.29	4.0343
		Average	4.4232

Figure 5.5 Time complexity and incremental ratio of the tested classes

Although the INN nets perform about the same as the traditional neural nets, their performances measured by time reduction ratio are very impressive. We witnessed a reduction of training time by about a factor of 4. In an online system setting, such an improvement is significant, and can enable the IRES system to respond to change much more quickly.

We can also tell that the nature of the incremented category does not have a significant effect on the time complexity of learning. Given the hypothesis that the problem nature was captured in the 25-outcome-class nets, we presume that most of the learning effort occurred during the training of those nets, resulting in similar performances in the INN training stage. However, the weak incremental classes do seem to have a slightly smaller reduction rate. This suggests to us that when the number of training cases gets big enough, the difference of time complexity among different incremental classes may get more significant.

5.2.3 Experiment Improvements

A statistical test is applied to the results to determine whether there is a difference in average precision and recall between the two sets of data (building from scratch vs. INN). The pooled t-test can calculate the average and standard deviation of both sets and determine whether the null hypothesis of two averages being the same is accepted.

Based on the t-tests result, average precision and recall rates in all six experiments are equal in two algorithms. This confirms our hypothesis that the INN reduces training time without reducing performance. Actual data of the statistical test is in Appendix A.

Another interesting improvement of the experiment would be to examine the impact of data size as well as number of outcomes on the performance of the INN algorithm. Since the IRES data set is too small, we have to come up with some simulation to work on this. Next section will review the experiment results for larger size data sets.

5.3 Experiments with Large Data Set

To further explore the impact of data set size as well as outcome class numbers on INN is necessary in order to generalize our findings. Unfortunately, the data set we have at hand is too small for this purpose. As a result, we created a data set with 2,000 records using the IRES data. Each instance in the original data set was repeated 10 times in the new data set. By doing this, we increased the data set size without alternating the statistical distribution of data. This section reports the experiment results using this set of data (All experiments below are tested by two pair t-test to guarantee statistical significance of the findings).

5.3.1 Impact of outcome class numbers on INN

Four sets of new experiments are carried out. Table 5.6 describes the scheme of these experiments.

We want to analyze the time reduction, precision as well as recall rated for all these experiments. If the input-hidden layer matrix captures the feature well, then the number of output classes wouldn't significantly affect above evaluation metrics.

Experiment Number	Original Number of Outcome Classes	New Number of Outcome Classes	Class Added
1	2	3	6
2	12	13	6
3	24	25	6

Table 5.6 Scheme of Four New Experiments

Surprisingly, the result shows that the number of outcome classes does have an impact on the learning time, although the performance is not affected.

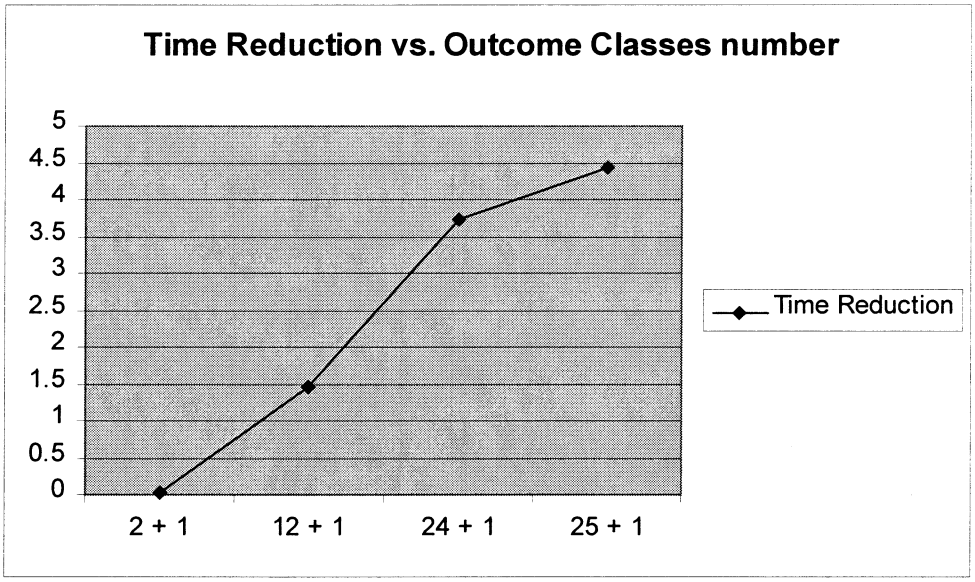


Figure 5.8 Impact of outcome class number on time reduction

The reason for the "time expansion" effect when outcome number change from 2 to 3 is due to the fact that the INN takes more training epochs than the net built from scratch to reach satisfactory performance.

The above results suggest that the hidden-output matrix must also be responsible for part of the pattern changes in the old classes. This is not an issue in the single outcome scenario, because no new pattern about the old classes appears.

In multiple outcome scenarios, however, new patterns of the old classes may be "imbedded" in the training instances containing the new outcome class. As a result, the hidden-output matrix has to capture these patterns. When the of number outcome classes is large, new cases containing new outcome class won't change the old patterns significantly, since they "distribute" the changes. For example, when there are already 25 classes, a case containing a new class may have 2^{25} possible ways of containing the old classes. As a result, chance of having an individual old class is small. However, when there are originally only two classes, the addition of a new class changes the pattern of the old classes drastically. The extra time that INN with smaller outcome class number spent is likely for capturing the changes in old classes, rather than learning the pattern of the new one.

As a conclusion, INN works best with large number of outcome classes.

5.3.2 Impact of data set size on INN

Will the size of the training data set affect the efficiency of INN? Below is a comparison between the same experiment carried out in 200 cases data set and 2000 cases data set:

Data Set	Precision	Recall	Time Reduction
200	76.80	72.18	4.42
2000	93.19	87.41	3.71

Table 5.7 Comparison of Performance in Data Sets of Different Sizes

Although there is an increase in performance and a decrease in time reduction factor, more experiments are needed for a final conclusion to be made. Further experiments are underway.

5. 4 Summary

The INN approach proposed was employed to solve a hypothetical incremental IRES problem using real world data set. The empirical experiments showed that the INN approach performed surprisingly well compare with the traditional approach. INN net has slightly lower precision rate than the traditional net, its recall rate is about the same, and sometimes better than, that of a traditional net. The training effort for the incremental procedure is smaller by about a factor of 4.

The nature of the incremental class does not seem to have an effect on the learning efficiency of the INN algorithm. Further statistical t-test has proven this.

When the number of outcome classes decreases, the time reduction rate also decreases. The extreme condition (two outcome classes are changed to three) even produces "time expansion". This is explained by the fact that the hidden-output layer in

the neural network is responsible for capturing extra changes in old outcome class patterns in a multiple outcome application. This seems to conflict with the theoretical foundation of this thesis: the input-to-hidden layer matrix serves as a feature detector. However, we also stated as a prerequisite that the new instances should NOT change old pattern significantly. When the new instances bring significant pattern changes to old classes, the input-to-layer matrix can no longer adapt to the new features. As a result, the hidden-to-output matrix has to compensate and take partial role in learning new patterns for the old classes.

In short, the INN algorithm can handle multiple outcome problems efficiently when the outcome class number is big. It reduces training time without losing performance.

CHAPTER 6

CONCLUSIONS

This chapter concludes the thesis, summarizing its contributions and making some suggestions for future research directions.

6. 1 Summary

The main objective of this thesis was to develop a new incremental neural network approach in order to apply it to applications set in an real-world, dynamically changing environment. After identifying the incremental learning problems associated with the limited learning ability of traditional neural networks, going over some literature reviews, and analyzing the specific IRES application, we proposed an incremental neural network (INN) technique based on the hidden layer activation approach designed by Ramani [Ramani 92]. This approach was tested to induce patient image retrieval knowledge from radiological image reading cases after new decision outcome class was added. The empirical experiment has shown that the INN approach performed surprisingly well compared to the traditional approach, and was able to reduce the training time by about a factor of 4.

6.2 Contributions of the Research

In order to build an active and adaptive IRES system, incremental learning techniques are needed. INN algorithm was applied in this research, and the potential contribution of it can be expressed from the perspective of both the radiological image retrieval application and the methodology (i.e., the INN network).

- The incremental learning problem is a big barrier for the IRES system to become active and adaptive. Radiologists' evolving image retrieval knowledge and changing needs for the knowledge system will make the algorithms designed in previous studies outdated very quickly. This research offered a potential technique to make the IRES system to be more adaptive to changes in the radiologists' needs.

- To the best of our knowledge, no past attempt has been made to build an incremental neural network to deal with changes in joint decision outcomes that are common in medical applications. This research has introduced a new data mining technique that can be applied to these challenging applications. Therefore, the research will shed some light on the theoretical underpinnings of the new technique and its applicability to a broad range of data domains.

6. 3 Suggestions for Future Research

Considering the increasing demand for incremental data mining in complex databases, much work in this area is still needed. The following areas are suggested for future research:

1. 200 cases were used in this study. This relatively small number of data set may not be able to reflect the impact of the incremental nature of new classes on the time reduction INN can achieve. In order to further investigate this, collection of more data is needed. Although a 2000 test case was included, more experiments need to be done in order to reveal the impact of data set size on INN performance.

2. Even though the result is encouraging, we want to have incremental learning algorithms that can produce rules that make sense to actual users. Continued research is needed in the area of incremental decision trees and other incremental data mining approach.

REFERENCES

[Agrawal 93] R. Agrawal, C. Faloutsos, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases", in *Proceedings of ACM SIGMOD*, pp. 207-216, 1993

[Agrawal 94] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases" in *Proceedings of 20th International Conference of Very Large Databases*, pp. 478- 499, 1994

[Burzzone 99] L. Burzzone, P. D. Fernandez and R. Cossu, "An incremental learning classifier for remote-sensing images," in IGARSS '99 Proceedings of IEEE International Geoscience and Remote Sensing Symposium,Vol. 5, pp. 2483 –2485, 1999

[Chen 96] M. Chen, J. Han and P. S. Yu, "Data Mining: An Overview from a Database Perspective", in *IEEE transaction on knowledge and data engineering*, Vol. 8, No. 6, pp. 866-883, 1996

[Cybenko 89] G. Cybenko, "Approximation by Superposition of a Sigmoidal Function," in *Mathematical Control, Signals, and Systems*, Vol 2, pp. 303 –314, 1989

[Ester 95] M. Ester, H. –P. Kriegel, and X. Xu, "Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification", in *Proceedings of Fourth International Symposium of Large Spatial Databases (SSD' 95)*, 1995

[Fahlman 90] S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture", in D. S. Touretzky (ed.), *Advances in neural information processing systems*, Vol. 2, pp. 524 – 532, 1990

[Fayyad 96] U. M. Fayyad, G. Piatetsky-Shapiro P. Smyth and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT press, 1996

[Fisher 87] D. Fisher, "Improving Inference Through Conceptual Clustering" in *Proceedings of AAAI Conference*, pp. 461-465, 1987

[Freudenheim 92] M Freudenheim, "Software Helps patients make crucial choices", *The New York Times*, October 14, 1992

[Gold 89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989

[Han 96a] J. Han and Y. Fu, "Exploration of the Power of Attribute-Oriented Induction in Data Mining," in *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT press, 1996

[Han 96b] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K.Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O.R. Zaiane, "DBMiner: A System for Mining Knowledge in Large Relational Databases," in *Proceedings of the International Conference in Data Mining and Knowledge Discovery (KDD' 96)*, pp. 250 – 255, 1996

[Hebert 99] J.-F. Hebert, M. Parizeau, and N. Ghazzali, "Cursive character detection using incremental learning," in *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pp.808–811,1999

[Hint 86] G. E. Hinton and T.J. Sejnowski, "Learning and Relearning in Boltzmann Machines," in *Parallel Distributed Processing,* Vol. 1, Cambridge, MA: MIT, 1986

[Hornik 89] Hornik, Kurt, Maxwell Stinchcombe and Halbert White, "Multilayer Feedforward Networks are Universal Approximators," in *Neural Networks* 2(5), pp. 359 – 366, 1989

[Hunt 66] E. Hunt, J. Martin, and P Stone, *Experiments in Induction*, New York, Academic Press, 1966

[Irie 88] Irie, Bunpei, and Sei Miyake, Capabilities of Three-layered Perceptrons," in *IEEE International Conference on Neural Networks*, Vol. I, pp. 161 – 172, 1988

[Jain 88] A. K. Jain and R. C Dubes, *Algorithms for Clustering Data*, Prentice Hall, 1988.

[Kishore 92] S. Kishore, "On-Demand Retrieval Paradigm," *in Proceedings of SPIE Medical Imaging IV: PACS Design and Evaluation,* pp149-157, 1992

[Kita 90] H. Kitano, "Empirical Studies on the Speed of Convergence of Neural Network Training Using Genetic Algorithms", in *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 785 – 795, 1990

[Kohon 90] T. Kohonen, "The Self-Organizing Map," in *Proceedings of the IEEE*, Vol. 78, No. 9, pp. 1464 – 1480, 1990.

[Lipp 87] R. P. Lippmann, "An Introduction to Computing with Neural Network," in *IEEE Acoustics Speech and Signal Processing Magazine*, Vol. 4, pp. 4 – 22, 1987

[Levin 90] K. Levin and R. Fielding, "Method to Pre-fetch Comparison Images in *Image Management and Communication Systems (IMCS)", in Proceedings of SPIE Medical Imaging IV: PACS Design and Evaluation,* Newport Beach, CA, pp. 270-274, 1990

[Lodder 91] H. Lodder, B. M. Poppel and A. R. Bakker, " Pre- fecthing: PACS Image Management Optimization Using HIS/ RIS Information" in *Proc. Of SPIE Medical Imaging: PACS Design and Evaluation,* pp. 227-233, 1991

[Looney 96] C. Looney, "Advances in Feedforward Neural Networks: Demystifying Knowledge Acquisition in Black Boxes," in *IEEE Transaction of Knowledge and Data Engineering*, pp. 211 – 226, 1996

[McClelland 94] J. L. McClelland, B. L. McNaughton and R. C. O'Railly, "Why there are Complementary Learning Systems in the Hippocampus and Neocortex: Insights from the successes and Failures of Connectionist Models of Learning and Memory," in *Technical Report: PDP.CNS.94.1* 1994

[McCloskey 89] M. McCloskey and N.J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problems," in *The psychology of learning and motivation*, Vol. 24, pp. 109 – 165, 1989

[Mchwz 92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin Heidelberg, 1992

[Nash 94] K. S. Nash, "Tufts implants decision support system – Red Brick's database improve data access for the HMO", *Computer world*, August 1, 1994

[Ng 94] R. Ng and J. Han, "Efficient and Effective Clustering Method for Spatial Data Mining", in *Proceedings of International Conference of Very Large Databases*, pp. 144- 155, 1994

[Park 95] J. S. Park, M. –S. Chen, and P. S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules", in *Proceedings of ACM SIGMOD*, pp. 175-186, 1995

[Pessa 94] E. Pessa and M.P. Pennaand, "Catastrophic Interference in Learning Process by Neural Networks," in *Proceedings of the ICANN'94*, 1994

[Piatetsky 91] G. Piatetsky-Shapiro and W. J. Frawley, in *Knowledge Discovery in Databases*, AAAI/MIT press, 1991

[Poggio 90] T. Poggio and F. Girosi, "Networks for Approximation and Learning," in *proceedings of IEEE*, Vol. 78, no. 9, pp. 1481 – 1497, 1990

[Quin 88] J. R. Quinlan, "An Empirical Comparison of Genetic and Decision-Tree Classifiers," in *Proceedings of the Fifth International Conference on Machine Learning*, pp. 135 – 141, 1988.

[Ramani 92] N. Ramani, "Incremental Learning Using Hidden Layer Activations – Tests On Fish Identification Data," in @$@!$#@!

[Rosen 62] F. Rosenblatt, *Principles of Neurondynamics: Perceptrons and the Theory of Brain Mechanisms*, New York: Spartan, 1962

[Rumel 86] D. E. Rumelhart, G. E. Hinton, and R. J Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1: Foundations, pp. 318 – 362, 1986

[Sheng 94a] O. Liu Sheng, C. Wei, M. Huang and P. Hu, "Machine Learning for Radiological Image Retrieval Knowledge," in *Proceedings of 8th Conference on Computer Assisted Radiology (SCAR-94),* Winston-Salem, North Carolina, pp. 360-367, 1994

[Sheng 94b] O. Liu Sheng, P. J. Hu, C. P. Wei, T. Ovitt and S. Akata,

"Acquisition and Evaluation of Knowledge-based Image Retrieval System," in *Proceedings of SPIE Medcial Imaging VIII: PACS Design and Evaluation*, Newport Beach, CA, pp. 714-725, 1994

[Sheng 94c] O. Liu Sheng, C. P. Wei and P. J. Hu, "Engineering Patient Image Retrieval Knowledge", in *Journal of Knowledge Engineering and Technology,* Vol. 7, No. 2, pp. 44-60, 1994

[Sheng 98] O. Liu Sheng, C. P. Wei and P. J. Hu, "Neural Network Learning for Intelligent Patient Image Retrieval," in IEEE Intelligent Systems, Vol. 13, No. 1, pp. 49 – 57, 1998

[Sej 86] T. Sejnowski and C. R. Rosenerg, "NETtalk: A Parallel Network that Learns to read Aloud", *Johns Hopkins Univ. Technical report JHU/EEC-86/01*, 1986

[Syed, 99] N. A.Syed, H. Liu and K.K. Sung, "Handling concept drifts in incremental learning with support vector machines", *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, Pages 317-321, August 15 - 18, 1999, San Diego,

[Villiers 92] J. de Villiers and E. Barnard, "Backpropagation Neural Nets with One and Two Hidden Layers," in *IEEE transactions on Neural Networks,* Vol. 4, No. 1, pp. 136 – 141, 1992

[Weiss 91] S. M. Weiss and C. A. Kulikowski, *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, Morgan Kaufman, 1991

[Widrow 90] B. Widrow, "30 Years of Adaptive Neural Networks: perceptron, Madaline, and Back Propagation", *Proc. IEEE*, Vol. 78, No.9, pp. 1415 – 1442, 1990

[Wilson 94] D.L. Wilson, D. Smith and B. Rice, "Intelligent Pre-fetch Strategies for Historical Images in a Large PACS," in *Proc. Of SPIE Medical Imaging IV: PACS Design and Evaluation,* Newport Beach, CA, pp. 112-123, 1994

[Zhou 99] Z. Zhou, S. Chen, and Z. Chen, "Mining typhoon knowledge with neural networks," in *Proceedings of 11th IEEE International*

Conference on Tools with Artificial Intelligence, pp. 325 –326, 1999

APPENDIX A. TRAINING RESULTS

Test 1: 25 classes → 26 classes, Class 6

(* refers to INN results, while no * refers to neural nets built from scratch. Used throughout this appendix)

Data Set	Precision	Recall	Time(s)	Precision*	Recall*	Time(s)*
1	87.92	71.8	1170.514	72.6	67.5	271.961
2	82.5	74.23	1119.751	68.5	60.8	267.865
3	69.58	70	942.694	65.83	63.33	197.865
4	82.5	70.5	1030.482	79.08	73.42	283.998
5	79.12	73.17	1022.411	77.92	68.17	229.21
Average	80.324	71.94	1057.1704	72.786	66.644	250.1798
Deviation	6.783294185	1.775795596	89.14655896	5.761135305	4.850528837	35.7331374

Two Sample t-test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision	Ha: Not equal to 0	0			
Input Column 2	Precision*	Confidence	0.95			
		Pooled Variance	TRUE			
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Precision	5	80.3240	6.78329	3.03358		
Precision*	5	72.7860	5.76114	2.57646		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
7.5380	3.98004	1.894	8.00	0.095	-1.6400	16.7160
Two Sample t-test for Recall						
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Recall	5	71.9400	1.77580	0.79416		
Recall*	5	66.6440	4.85053	2.16922		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
5.2960	2.31003	2.293	8.00	0.051	-0.0309	10.6229

Test 2: 25 classes → 26 classes, Class 7

Data Set	Precision	Recall	Time(s)	Precision	Recall	Time(s)
1	87.92	71.8	1170.514	80.67	76.67	233.33
2	82.5	74.23	1119.751	86.67	79.01	205.241
3	69.58	70	942.694	68.75	70	284.209
4	82.5	70.5	1030.482	81.33	75.5	204.312
5	79.12	73.17	1022.411	79.17	74.41	202.653
Average	80.324	71.94	1057.1704	79.318	75.118	225.949
Deviation	6.783294185	1.775795596	89.14655896	6.550428994	3.331571701	34.95851617

Two Sample t-test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision	Ha: Not equal to 0	0			
Input Column 2	Precision*	Confidence	0.95			
		Pooled Variance	TRUE			
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Precision	5	80.3240	6.78329	3.03358		
Precision*	5	79.3180	6.55043	2.92944		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
1.0060	4.21714	0.239	8.00	0.817	-8.7187	10.7307
Two Sample t-test for Recall						
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Recall	5	71.9400	1.77580	0.79416		
Recall*	5	75.1180	3.33157	1.48992		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
-3.1780	1.68836	-1.882	8.00	0.097	-7.0714	0.7154

Test 3: 25 classes → 26 classes, Class 11

Data Set	Precision	Recall	Time(s)	Precision*	Recall*	Time(s)*
1	87.92	71.8	1170.514	77.92	68.33	273.944
2	82.5	74.23	1119.751	77.83	68.48	267.665
3	69.58	70	942.694	77.08	78.17	162.824
4	82.5	70.5	1030.482	78.67	77.58	338.036
5	79.12	73.17	1022.411	72.5	68.33	146.901
Average	80.324	71.94	1057.1704	76.8	72.178	237.874
deviation	6.783294185	1.775795596	89.14655896	2.468835758	5.205167625	80.82320838

Two Sample t-test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision	Ha: Not equal to 0	0			
Input Column 2	Precision*	Confidence	0.95			
		Pooled Variance	TRUE			
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Precision	5	80.3240	6.78329	3.03358		
Precision*	5	76.8000	2.46884	1.10410		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
3.5240	3.22826	1.092	8.00	0.307	-3.9204	10.9684
Two Sample t-test for Recall						
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Recall	5	71.9400	1.77580	0.79416		
Recall*	5	72.1780	5.20517	2.32782		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
-0.2380	2.45956	-0.097	8.00	0.925	-5.9098	5.4338

Test 4: 25 classes → 26 classes, Class 16

Data Set	Precision	Recall	Time(s)	Precision*	Recall*	Time(s)*
1	87.92	71.8	1170.514	80	79.17	300.793
2	82.5	74.23	1119.751	76.67	74.19	196.713
3	69.58	70	942.694	67.5	67.5	428.035
4	82.5	70.5	1030.482	81.65	72.58	196.543
5	79.12	73.17	1022.411	82.08	73.17	188.14
average	80.324	71.94	1057.1704	77.58	73.322	262.0448
deviation	6.783294185	1.775795596	89.14655896	6.022993442	4.165989678	103.7723697

Two Sample t-test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision	Ha: Not equal to 0	0			
Input Column 2	Precision*	Confidence	0.95			
		Pooled Variance	TRUE			
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Precision	5	80.3240	6.78329	3.03358		
Precision*	5	77.5800	6.02299	2.69356		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
2.7440	4.05683	0.676	8.00	0.518	-6.6111	12.0991
Two Sample t-test for Recall						
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Recall	5	71.9400	1.77580	0.79416		
Recall*	5	73.3220	4.16599	1.86309		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
-1.3820	2.02529	-0.682	8.00	0.514	-6.0523	3.2883

Test 5: 25 classes → 26 classes, Class 2

Data Set		Precision: Scratch	Precision:INN	Recall: S	Recall: INN	Time:S	Time:INN
	1	87.92	75.08	71.8	81.25	1170.514	170.695
	2	82.5	79.33	74.23	81.73	1119.751	159.65
	3	69.58	68.75	70	70	942.694	284.209
	4	82.5	81.33	70.5	75.5	1030.482	263.312
	5	79.12	79.17	73.17	74.41	1022.411	198.653
Average		80.324	76.732	71.94	76.578	1057.1704	215.303
Deviation		6.783294185	5.006907229	1.775795596	4.937182395	89.14655896	55.7163653

Two Sample t-test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision: Scratch	Ha: Not equal to 0	0			
Input Column 2	Precision:INN	Confidence	0.95			
		Pooled Variance	TRUE			
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Precision: Scratch	5	80.3240	6.78329	3.03358		
Precision:INN	5	76.7320	5.00691	2.23916		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
3.5920	3.77047	0.953	8.00	0.369	-5.1027	12.2867
Two Sample t-test for Precision						
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Recall: S	5	71.9400	1.77580	0.79416		
Recall: INN	5	76.5780	4.93718	2.20798		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
-4.6380	2.34645	-1.977	8.00	0.083	-10.0489	0.7729

Test 6: 25 classes → 26 classes, Class 3

Data Set	Precision	Recall	Time(s)	Precision*	Recall*	Time(s)*
1	87.92	71.8	1170.514	77.58	72.92	270.82
2	82.5	74.23	1119.751	76.67	78.51	265.572
3	69.58	70	942.694	73.33	73.33	335.343
4	82.5	70.5	1030.482	76.58	75.08	195.772
5	79.12	73.17	1022.411	73	76.92	177.425
Average	80.324	71.94	1057.1704	75.432	75.352	248.9864
Deviation	6.783294185	1.775795596	89.14655896	2.109352981	2.372165677	63.56503936

Two Sample t-test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision	Ha: Not equal to 0	0			
Input Column 2	Precision*	Confidence	0.95			
		Pooled Variance	TRUE			
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Precision	5	80.3240	6.78329	3.03358		
Precision*	5	75.4320	2.10935	0.94333		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
4.8920	3.17687	1.540	8.00	0.162	-2.4339	12.2179
Two Sample t-test for Recall						
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Recall	5	71.9400	1.77580	0.79416		
Recall*	5	75.3520	2.37217	1.06086		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
-3.4120	1.32519	-2.575	8.00	0.033	-6.4679	-0.3561

Test 7. 2 classes → 3 classes, class 6

Data Set	Precision	Recall	Time(s)	Precision*	Recall*	Time(s)*
1	96.06	94.23	2.904	90.73	92.81	73.976
2	94.88	93.17	1.943	94.14	92.65	78.994
3	93.41	94.96	3.235	92.35	91.7	89.137
4	96.58	94.26	2.944	90.57	92.98	82.487
average	95.2325	94.155	2.7565	91.9475	92.535	81.1485
deviation	1.407867773	0.738218576	0.562037069	1.668220109	0.572741943	6.369048621

Two-Pair t-Test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision	Ha: Not equal to 0	0			
Input Column 2	Precision*	Confidence	0.95			
		Pooled Variance	TRUE			
Descriptive Statistics						
	N	***Mean***	***Std. Dev.***	***Std. Err.***		
Precision	4	95.2325	1.40787	0.70393		
Precision*	4	91.9475	1.66822	0.83411		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
3.2850	1.09145	3.010	6.00	0.024	0.6143	5.9557
Two-Pair t-Test for Recall						
Descriptive Statistics						
	N	***Mean***	***Std. Dev.***	***Std. Err.***		
Recall	4	94.1550	0.73822	0.36911		
Recall*	4	92.5350	0.57274	0.28637		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
1.6200	0.46717	3.468	6.00	0.013	0.4769	2.7631

Test 8. 23 classes → 24 classes, class 6

Data Set	Precision	Recall	Time(s)	Precision*	Recall*	Time(s)*
1	94.22	85.65	3015.466	94.84	84.09	803.165
2	94.08	88.04	2982.939	95.08	88.27	788.023
3	95.13	91.21	2971.232	95.73	89.82	788.793
4	95.52	89.35	2821.057	96.16	87.11	788.344
average	94.7375	88.5625	2947.6735	95.4525	87.3225	792.08125
deviation	0.699160211	2.337026244	86.46043323	0.603179078	2.424147617	7.39591177

Two-Pair t-Test for Precision						
Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0.00000			
Input Column 1	Precision	Ha: Not equal to 0	0.00000			
Input Column 2	Precision*	**Confidence**	0.95			
		Pooled Variance	**TRUE**			
	Descriptive Statistics					
	N	***Mean***	***Std. Dev.***	***Std. Err.***		
Precision	4	94.7375	0.69916	0.34958		
Precision*	4	95.4525	0.60318	0.30159		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
-0.7150	0.46170	-1.549	6.00	0.172	-1.8447	0.4147
	Descriptive Statistics					
	N	**Mean**	**Std. Dev.**	**Std. Err.**		
Recall	4	88.5625	2.33703	1.16851		
Recall*	4	87.3225	2.42415	1.21207		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
1.2400	1.68361	0.737	6.00	0.489	**-2.8797**	**5.3597**

Test 9. 12 classes → 13 classes, class 6

Data Set	Precision	Recall	Time(s)	Precision*	Recall*	Time(s)*
1	95.12	95.76	1423.46	97.14	93.64	932.04
2	94.23	93.24	1343.25	96.81	94.24	934.835
3	96.94	94.56	1468.94	96.71	92.9	932.68
4	94.33	95.18	1268.67	94.71	94.67	969.634
average	95.155	94.685	1376.08	96.3425	93.8625	942.29725
deviation	1.2548174	1.08078675	88.4762	1.10373	0.768218	18.263678

Two-Pair t-Test for Precision

Parameters						
Analysis	2 Sample t	Ho: Mean Diff. = 0	0			
Input Column 1	Precision	Ha: Not equal to 0	0			
Input Column 2	Precision*	Confidence	0.95			
		Pooled Variance	TRUE			
	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Precision	4	95.1550	1.25482	0.62741		
Precision*	4	96.3425	1.10373	0.55187		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
-1.1875	0.83558	-1.421	6.00	0.205	-3.2321	0.8571

Two-Pair t-Test for Recall

	Descriptive Statistics					
	N	Mean	Std. Dev.	Std. Err.		
Recall	4	94.6850	1.08079	0.54039		
Recall*	4	93.8625	0.76822	0.38411		
	t-Test Analysis					
Mean Diff.	Std. Err.	t	df	p-value	lower 95%	upper 95%
0.8225	0.66300	1.241	6.00	0.261	-0.7998	2.4448

APPENDIX B. CODES

```
/** Lin Lin, INN Project Version 1.0
*   class DataAnalyzer is a helper class that reads in a data file and
analyze the distribution of its
*   data in output space
*/



import java.util.*;
import java.io.*;

public class DataAnalyzer
{
    private TextReader myReader;         //Read in the file
    public Vector myData;                //Used to store records(examples)
    from the original file
    public int inputNum, outputNum;
    public int[]  outputArray;
    public int[]  randomTest;            //USed to store randomly created
    index for choosing testing examples

    public int[] category;               //Hold number of appearance for each
    output category



    public DataAnalyzer(String filename)
    {
        myReader = new TextReader(filename);
        myData = new Vector();
    }

    public void analyze()
    {
        readFile();
        count();
    }



    public void readFile()
    //Read a file and store all the examples in vector myData
    {
        int lineNum = 0;
        int count = 0;
        String curr = "";

        while(myReader.ready())
        {
            if(lineNum == 0)
            {
                inputNum =
                Integer.parseInt(myReader.readLine().substring(0,5).trim());
                lineNum++;
            }

            else if(lineNum == 1)
            {
                outputNum = Integer.parseInt(myReader.readLine().substring(0,
                5).trim());
                lineNum++;
                outputArray = new int[outputNum];
                category = new int[outputNum];
            }

            else // Begin to read data
            {
```

```
            while(count <1)
            {
                if((myReader.readWord()).equals(","))
                {
                    count++;
                }
            }

            String temp = myReader.readWord();

            if(temp.equals(","))
                count++;
            else
            {
                curr += temp;
            }

            if(count == 2)
            {
                myData.addElement(curr);
                count = 0;
                curr = "";
            }
        }
    }
}


    public void count()
    {
        for(int i= 0; i < myData.size(); i++)
        {

            if((((String)
            myData.elementAt(i)).substring(0,1)).equals("1"))
                category[0] += 1;

            if((((String)
            myData.elementAt(i)).substring(1,2)).equals("1"))
                category[1] += 1;

            if((((String)
            myData.elementAt(i)).substring(2,3)).equals("1"))
                category[2] += 1;

            if((((String)
            myData.elementAt(i)).substring(3,4)).equals("1"))
                category[3] += 1;

            if((((String)
            myData.elementAt(i)).substring(4,5)).equals("1"))
                category[4] += 1;

            if((((String)
            myData.elementAt(i)).substring(5,6)).equals("1"))
                category[5] += 1;

            if((((String)
            myData.elementAt(i)).substring(6,7)).equals("1"))
                category[6] += 1;

            if((((String)
            myData.elementAt(i)).substring(7,8)).equals("1"))
                category[7] += 1;

            if((((String)
            myData.elementAt(i)).substring(8,9)).equals("1"))
```

```
                category[8] += 1;

            if((((String)
            myData.elementAt(i)).substring(9,10)).equals("1"))
                category[9] += 1;
            if((((String)myData.elementAt(i)).substring(10,11)).equals(
            "1"))
                category[10] += 1;
            if((((String)myData.elementAt(i)).substring(11,12)).equals(
            "1"))
                category[11] += 1;
            if((((String)myData.elementAt(i)).substring(12,13)).equals(
            "1"))
                category[12] += 1;
            if((((String)myData.elementAt(i)).substring(13,14)).equals(
            "1"))
                category[13] += 1;
            if((((String)myData.elementAt(i)).substring(14,15)).equals(
            "1"))
                category[14] += 1;
            if((((String)myData.elementAt(i)).substring(15,16)).equals(
            "1"))
                category[15] += 1;
            if((((String)myData.elementAt(i)).substring(16,17)).equals(
            "1"))
                category[16] += 1;
            if((((String)myData.elementAt(i)).substring(17,18)).equals(
            "1"))
                category[17] += 1;
            if((((String)myData.elementAt(i)).substring(18,19)).equals(
            "1"))
                category[18] += 1;
            if((((String)myData.elementAt(i)).substring(19,20)).equals(
            "1"))
                category[19] += 1;
            if((((String)myData.elementAt(i)).substring(20,21)).equals(
            "1"))
                category[20] += 1;
            if((((String)myData.elementAt(i)).substring(21,22)).equals(
            "1"))
                category[21] += 1;
            if((((String)myData.elementAt(i)).substring(22,23)).equals(
            "1"))
                category[22] += 1;
            if((((String)myData.elementAt(i)).substring(23,24)).equals(
            "1"))
                category[23] += 1;
            if((((String)myData.elementAt(i)).substring(24,25)).equals(
            "1"))
                category[24] += 1;
            if((((String)myData.elementAt(i)).substring(25,26)).equals(
            "1"))
                category[25] += 1;

        }
    }



public static void main(String[] args)
{
    DataAnalyzer x = new DataAnalyzer("testData1.txt");
    x.analyze();

    for(int i=0; i < x.myData.size(); i++)
    {
```

```
            System.out.println(x.myData.elementAt(i));
        }
        System.out.println("my input is : " + x.inputNum + ", my output: "+
        x.outputNum);
        System.out.println(" The size of the data is: " + x.myData.size());
        for(int i=0;i<x.outputNum; i++)
            System.out.println(x.category[i]);
    }
}
```

```
/** Lin Lin, INN Project Version 1.0
*   class DataFile read a data file and construct a Vector to store all the
training samples
*   it is used as the interface between the neural network and the data files
*/


import java.util.*;
import java.io.*;

public class DataFile
{
    private TextReader myReader;        //Read in the file
public int inputNum, outputNum;
    public int[] inputData, outputData;

    public DataFile(String filename)
    {
        myReader = new TextReader(filename);
        myData = new Vector();
    }

    public Vector getData()
    {
        readFile();
        return myData;
    }

    public void readFile()
    {
        int lineNum = 0;
        String curr = "";
        int count = 0;

        while(myReader.ready())
        {
            if(lineNum == 0)
            {
                inputNum =
                Integer.parseInt(myReader.readLine().substring(0,5).trim());
                lineNum++;
            }

            else if(lineNum == 1)
            {
                outputNum = Integer.parseInt(myReader.readLine().substring(0,
                5).trim());
                inputData = new int[inputNum];
                outputData = new int[outputNum];
                lineNum++;
            }

            else  // Begin to read data
            {
                String temp = "";

                if(count==0)
                {
                     for(int i=0; i<inputNum; i++)
                         inputData[i] =
                         Integer.parseInt(myReader.readWord());
                     myData.addElement(inputData);
                     inputData = new int[inputNum];
                     temp = myReader.readWord();
                 }

                if(temp.equals(","))
```

```
            {
                count++;
                temp = "";
            }

            if(count==1)
            {
                for(int i=0; i<outputNum; i++)
                     outputData[i] = Integer.parseInt(myReader.readWord
                     ());

                 myData.addElement(outputData);
                 outputData = new int[outputNum];
                 temp = myReader.readWord();
             }

            if(temp.equals(","))
            {
                count++;
                temp = "";
            }

            if(count == 2)
                count = 0;
        }
    }
}


public void removeData(String fileName)
{
    PrintWriter out = null;

    try
    {
         out = new PrintWriter(new FileWriter(fileName));
    }
    catch (IOException e)
    {
        System.out.println("Can't open output file '" + fileName +
        "', exiting");
        System.exit(1);
    }

    out.println( inputNum + "                    ");
    out.println( (outputNum-1) + "                  ");

    for(int i= 1; i < myData.size(); i+=2)
    {
        if( ((int[])myData.elementAt(i))[9] == 1)
        System.out.println("Aha");

        else
        {
            for(int j=0; j< inputNum; j++)
            {
                out.print( " " + ((int[])myData.elementAt(i-1))[j] );
            }

            out.print(" ,");

            for(int j=0; j< outputNum ; j++)
            {
                if(j!= 1)
                   out.print( " " + ((int[])myData.elementAt(i))[j] );
            }

            out.println(" ,");
```

```
        }
    }
    out.close();
}


public static void main(String[] args)
{
    DataFile x = new DataFile("Lack5.txt");
    Vector y = x.getData();
    x.removeData("Lack5_10.txt");
    for(int i=0; i < y.size(); i++)
    {
        if(i%2 == 0)
        {
            for(int j = 0; j< x.inputNum; j++)
                System.out.print(((int[])y.elementAt(i))[j]+"   ");
            System.out.println();
            System.out.println("HAHA");
        }

        else
        {
            for(int j = 0; j< x.outputNum ; j++)
                System.out.print(((int[])y.elementAt(i))[j]+"    ");
            System.out.println();
        }

    }
    System.out.println("my input is : " + x.inputNum + ", my output: "+
    x.outputNum);
    System.out.println(" The size of the data is: " + x.myData.size());
}
public Vector myData;                //Used to store records(examples)
from the original file
}
```

```
/** Lin Lin, INN Project Version 1.0
*   class FileOrganizer is a helper class that reads in a data file and
randomly split it up into
*   one test file and one training file
*/



import java.util.*;
import java.io.*;

public class FileOrganizer
{
    private TextReader myReader;        //Read in the file
    public Vector myData;               //Used to store records(examples)
    from the original file
    public int inputNum, outputNum;
    public int[]  randomTest;           //USed to store randomly created
    index for choosing testing examples


    public FileOrganizer(String filename)
    {
        myReader = new TextReader(filename);
        myData = new Vector();
    }

    public void organize()
    {
        readFile();
        shuffle();
        writeFile();
    }



    public void readFile()
    //Read a file and store all the examples in vector myData
    {
        int lineNum = 0;
        String curr = "";
        int count = 0;

        while(myReader.ready())
        {
            if(lineNum == 0)
            {
                inputNum =
                Integer.parseInt(myReader.readLine().substring(0,5).trim());
                lineNum++;
            }

            else if(lineNum == 1)
            {
                outputNum = Integer.parseInt(myReader.readLine().substring(0,
                5).trim());
                lineNum++;
            }

            else // Begin to read data
            {
                String temp = myReader.readWord();
                curr += " " + temp;

                if(temp.equals(","))
                    count++;

                if(count == 2)
```

```
            {
                myData.addElement(curr);
                count = 0;
                curr = "";
            }
        }
    }
}


public void shuffle()
{
    //int testSize = (int)(myData.size() * 0.1);
    int testSize = (int)(myData.size() * 0.25);
    randomTest = new int[testSize];

    for(int i=0; i< testSize; i++)
    {
        randomTest[i] = (int)(myData.size() *Math.random());
        System.out.println(randomTest[i]);
    }

}

public void  writeFile()
{
    PrintWriter out1 = null;
    PrintWriter out2 = null;
    int out1Num = 0;
    int out2Num = 0;

    try
    {
        out1 = new PrintWriter(new FileWriter("Mad-1Test.txt"));
        out2 = new PrintWriter(new FileWriter("Mad-1Train.txt"));
    }
    catch (IOException e)
    {
         System.out.println("Can't open output file , exiting");
         System.exit(1);
    }

    out1.print(inputNum+"              " + "\n");
    out2.print(inputNum+"              " + "\n");
    out1.print(outputNum+"              " + "\n");
    out2.print(outputNum+"              " + "\n");

    for(int i = 0; i < myData.size(); i++)
    {
        boolean inTest = false;

        for(int j = 0; j<(int)(myData.size() * 0.25); j++)
        {
            if(i == randomTest[j])
            {
                out1.print(myData.elementAt(i));
                out1.print("\n");
                inTest = true;
                out1Num++;
            }
        }

        if(!inTest)
        {
            out2.print(myData.elementAt(i));
            out2.print("\n");
            out2Num++;
        }
```

```
        }

        out1.close();
        out2.close();
        System.out.println("Training data: "+out2Num + " Testing Data: "+
        out1Num);
    }


    public static void main(String[] args)
    {
        FileOrganizer x = new FileOrganizer("Mad-1.txt");
        x.organize();
        /*for(int i=0; i < y.size(); i++)
        {
            System.out.println(y.elementAt(i));
        }*/
        System.out.println("my input is : " + x.inputNum + ", my output: "+
        x.outputNum);
        System.out.println(" The size of the data is: " + x.myData.size());
    }
}
```

```
/** Lin Lin, INN Program Version 1.0
*
*   Class NNfile : parses neural network input files storing network
parameters.
*                  three - layer architecture is assumed
*
*   public NNfile()
*       Construct a "empty" net
*
*   public NNfile(String input_file)
*       Read from the file "input_file" and get the number of nodes for each
layer. If weights are
*       stored in the file, they also get read into the NNfile object
*
*   public void getWeights(TextReader in)
*       This method handles the details of loading the weight matrix into the
object
*
*   public float getW1(int i, int j)
*       Return a certain hidden layer weight
*
*   public float getW2(int i, int j)
*       Return a certain output layer weight
*
*   public static void saveFile( String fileName, Neural target)
*       A utility method used by Neural classes to save the trained structure
of the network
*
*/
import java.io.*;
import java.util.*;


public class NNfile {

  public int numInput, numHidden, numOutput; //Number of input, hidden, and
  output layers
  public boolean weightFlag;                 //Specify whether weight is also
  stored in file
  public float[][] inputWeights;
  public float[][] hiddenWeights;

  //Default constructor
  public NNfile()
  {
    numInput=numHidden=numOutput=0;
    weightFlag = false;
  }

  //Using a file name to construct a NNFile
  public NNfile(String input_file)
  {

    TextReader in = null;
    weightFlag = false;

    try
    {
      in = new TextReader(input_file);
    }
    catch (Exception E) { System.out.println("can not open file " +
    input_file);}

    try
    {
      if (in != null)  readFile(in);
    }
    catch (Exception E) { System.out.println("can not process file");}
```

```
  System.out.println("Done with ReadFile.");

}

public void readFile(TextReader in)
{
  String temp = "";
  int lineNumber = 0;
  int weightCount = 0;

  while(in.ready())
  {

      temp = in.readLine();

      lineNumber++;

      switch(lineNumber)
      {
          case 1:  numInput = Integer.parseInt(temp.substring(0,9).trim());
                   break;

          case 2:  numHidden= Integer.parseInt(temp.substring(0,9).trim());
                   break;

          case 3:  numOutput= Integer.parseInt(temp.substring(0,9).trim());
                   break;

          default:
      }

      if(lineNumber == 4) // comes to the forth line
      {
          if(temp.equals("No weights"))
          {
              weightFlag = false;
              return;
          }
          else
              weightFlag = true;

      }

      if(lineNumber == 4)
          getWeights(in);

  }
}


public void getWeights(TextReader in)
{
    inputWeights = new float[numInput][numHidden];
    hiddenWeights= new float[numHidden][numOutput];
    boolean isOut = false;
    int counter = 0;
    String temp = "";

    while(in.ready())
    {
        //System.out.println(isOut);
        if(!isOut)
        {
            temp = in.readWord();
            if(!temp.equals(","))
            {
              //System.out.println("Storing "+ counter / numHidden + "   " +
```

```
                counter % numHidden + "  "+ Float.parseFloat(temp) );
                inputWeights[counter / numHidden][counter % numHidden] =
                Float.parseFloat(temp);
                counter++;
            }
            else
            {
                //System.out.println("I am Here !!");
                counter = 0;
                isOut = true;
            }
        }

        else
        {
            temp = in.readWord();
            //System.out.println("Storing "+ counter / numOutput + "  " +
            counter % numOutput +"  "+ Float.parseFloat(temp) );
            hiddenWeights[counter / numOutput][counter % numOutput] =
            Float.parseFloat(temp);
            counter++;
        }
    }
}


public float getW1(int i, int j)
{
    //System.out.println( i + " " + j + inputWeights[i][j]);
    return inputWeights[i][j];
}


public float getW2(int i, int j)
{
    return hiddenWeights[i][j];
}


public static void saveFile( String fileName, Neural target)
{
    PrintWriter out = null;

    try
    {
      out = new PrintWriter(new FileWriter(fileName));
    }
    catch (IOException e)
    {
      System.out.println("Can't open output file '" + fileName +
      "', exiting");
      System.exit(1);
    }

    out.println(target.NumInputs + "                    " +
    " # of Input nodes");
    out.println(target.NumHidden + "                    " +
    " # of Hidden nodes");
    out.println(target.NumOutputs + "                    " +
    " # of Output nodes");

    //Will assume that weights are created
    out.println("Weights:");

    //Write the input - hidden weight matrix
    for (int i=0; i < target.NumInputs; i++)
```

```
    {
        for (int h=0; h<target.NumHidden; h++)
        {
           out.print(target.W1[i][h] + "  ");
        }
    }

    out.print("  ,  ");

    for (int h=0; h<target.NumHidden; h++)
    {
        for (int o=0; o<target.NumOutputs; o++)
        {
            out.print(target.W2[h][o] + "  ");
        }
    }
    out.close();
  }


public static void saveINN( String fileName, INN target)
 {
    PrintWriter out = null;

    try
    {
      out = new PrintWriter(new FileWriter(fileName));
    }
    catch (IOException e)
    {
      System.out.println("Can't open output file '" + fileName +
      "', exiting");
      System.exit(1);
    }

    out.println(target.NumInputs + "                    " +
    " # of Input nodes");
    out.println(target.NumHidden + "                    " +
    " # of Hidden nodes");
    out.println(target.NumOutputs + "                     " +
    " # of Output nodes");

    //Will assume that weights are created
    out.println("Weights:");

    //Write the input - hidden weight matrix
    for (int i=0; i < target.NumInputs; i++)
    {
        for (int h=0; h<target.NumHidden; h++)
        {
           out.print(target.W1[i][h] + "  ");
        }
    }

    out.print("  ,  ");

    for (int h=0; h<target.NumHidden; h++)
    {
        for (int o=0; o<target.NumOutputs; o++)
        {
            out.print(target.W2[h][o] + "  ");
        }
    }
    out.close();
  }

  /*public static void saveTwoLayerNNFile( String fileName, TwoLayerNN
  target)
```

```
    {
     PrintWriter out = null;

     try
     {
       out = new PrintWriter(new FileWriter(fileName));
     }
     catch (IOException e)
     {
       System.out.println("Can't open output file '" + fileName + "',
       exiting");
       System.exit(1);
     }

     out.println(target.NumInputs + "                    " + " # of Input
     nodes");
     out.println(target.NumOutputs + "                     " + " # of Output
     nodes");

     //Will assume that weights are created
     out.println("Weights:");

     //Write the input - hidden weight matrix
     for (int i=0; i < target.NumInputs; i++)
         for (int o=0; o<target.NumOutputs; o++)
            out.print(o.W[i][o] + "   ");

     out.close();
   }*/

}
```

```
import java.util.*;

class Neural

{
   protected int NumHidden;
   /**********************Data about neural net
   characteristics*********************/

   protected int NumInputs;
   protected int NumOutputs;

   protected boolean WeightsFlag;
   protected float[][] W1;
   protected float[][] W2;

   public float[] Inputs;
   protected float[] Hidden;
   public float[] Outputs;
   public int[] outData;      //The output in training data

   protected float[] output_errors;

   protected float[] totalError;

   protected Vector data;

   public NNfile neuralFile=null;
   public DataFile trainSet=null;


   /************************Data about training
   parameters*****************************/

   public float threshold;  //Used for testing
   public float lRate;      //learning rate
   public float momentum;   //momentum


   Neural()
   {
     NumInputs = NumHidden = NumOutputs = 0;
   }


   Neural(String NNfile_name, String dataFile_name, float thresh,float lRate,
   float momentum)
   {
     // Get  a file  and construct a net

     /********** First Step: Construct the framework of the neural net
     **************/

     neuralFile = new NNfile(NNfile_name);
     NumInputs = neuralFile.numInput;
     NumHidden = neuralFile.numHidden;
     NumOutputs = neuralFile.numOutput;
     WeightsFlag= neuralFile.weightFlag;


     //Set up all the training parameters
     threshold = thresh;
     this.lRate = lRate;
     this.momentum = momentum;


     Inputs = new float[NumInputs];
```

```
    Hidden = new float[NumHidden];
    Outputs = new float[NumOutputs];
    outData = new int[NumOutputs];
    W1 = new float[NumInputs][NumHidden];
    W2 = new float[NumHidden][NumOutputs];

    //Get the training or testing data
    trainSet = new DataFile(dataFile_name);
    data = trainSet.getData();

    output_errors = new float[NumOutputs];
    hidden_errors = new float[NumHidden];
    totalError = new float[data.size()/2];
    System.out.println("The errors are" + data.size()/2);

    // Retrieve the weight values from the NNfile object:
    initWeights();
}

  public void initWeights()
  {
      if (WeightsFlag)
           {
             for (int i=0; i<NumInputs; i++)
                   for (int h=0; h<NumHidden; h++)
                   W1[i][h] = neuralFile.getW1(i, h);

             for (int h=0; h<NumHidden; h++)
                for (int o=0; o<NumOutputs; o++)
                  W2[h][o] = neuralFile.getW2(h, o);
           }
           else
      randomizeWeights();
  }


  public void randomizeWeights()
  {
     // Randomize weights here:

     // 64 inputs nodes with value of 0 or 1, the summed weight aims at [-1,
     1] so that the
     // hidden nodes will be about 0.5
     for (int ii=0; ii<NumInputs; ii++)
        for (int hh=0; hh<NumHidden; hh++)
             W1[ii][hh] =  0.01f * (float)Math.random();



     for (int hh=0; hh<NumHidden; hh++)
        for (int oo=0; oo<NumOutputs; oo++)
             //W2[hh][oo] = (float)Math.random() - 0.5f;
             {
                 if(Math.random() >= 0.5)
                   W2[hh][oo] = 1;
                 else
                   W2[hh][oo] = -1;
             }
  }


  public float train()
  {

      //System.out.println("Learning rate is " + lRate);
      for (int example = 0; example < data.size(); example+=2 )
      {
          //System.out.println("example number " + example);
```

```
        // zero out error arrays:
        for (int h=0; h<NumHidden; h++)
             hidden_errors[h] = 0.0f;
        for (int o=0; o<NumOutputs; o++)
             output_errors[o] = 0.0f;

        // copy the input values:
        for (int i=0; i<NumInputs; i++)
           {
               Inputs[i] = (float)(((int[])data.elementAt(example))[i] );
               //System.out.print(Inputs[i]);
           }

        //System.out.println();

        //copy the training output
        for (int i=0; i<NumOutputs; i++)
            {
                outData[i] =((int[])data.elementAt(example+1))[i];
                //System.out.print(outData[i]);
            }

        //System.out.println();

        forwardPass();


       //System.out.println("The hidden is:");

            /*for(int i=0; i< NumHidden; i++)
                 System.out.print( Hidden[i] + "   ");

            System.out.println();
            System.out.println("The output is:");

            for(int i=0; i< NumOutputs; i++)
                 System.out.print( Outputs[i] + "   ");
            System.out.println(); */

         totalError[example / 2] = weightAdjust();

         //System.out.println(totalError[example / 2]);
      }

    float error = 0.0f;

    for(int i = 0; i < data.size() / 2; i++)
          error += totalError[i];

    error /= (data.size()/2);
    return error;
}


public void test(String testFile)
{

     //Used to store the result
     int result[] = new int[NumOutputs];
     float[] precision = new float[data.size()/2];
     float[] recall = new float[data.size()/2];
     float totalError  = 0.0f;
     int correctNum = 0;
     int trueNum = 0, machineNum = 0;

     //Get the testing file
```

```
trainSet = new DataFile(testFile);
data = trainSet.getData();

for (int example = 0; example < data.size(); example+=2 )
{
        // copy the input values:
        for (int i=0; i<NumInputs; i++)
        {
           Inputs[i] = (float)(((int[])data.elementAt(example))[i] );
           //System.out.print(Inputs[i]);
        }

        //System.out.println();

        //copy the training output
        for (int i=0; i<NumOutputs; i++)
            {
                outData[i] =((int[])data.elementAt(example+1))[i];
                //System.out.print(outData[i]);
            }

        //System.out.println();

        forwardPass();

        for(int i =0; i<NumOutputs; i++)
             System.out.print(outData[i]);
        System.out.println();

        for(int i =0; i<NumOutputs; i++)
        {
             //System.out.print(Outputs[i] + "-->");
             if(Outputs[i] >= threshold)
               result[i] = 1;
             else
               result[i] = 0;
             System.out.print(result[i]);
        }
        System.out.println();
        System.out.println();

        //Calculate the accuracy
        for(int i =0; i<NumOutputs; i++)
        {
            if(outData[i] == result[i] && result[i]==1)
            {
                correctNum++;
                machineNum++;
                trueNum++;
            }

            else
            {
                if(result[i] == 1)
                  machineNum++;

                if(outData[i]==1)
                  trueNum++;
            }
        }

        if(machineNum != 0)
             precision[example/2] = 1.0f * correctNum / machineNum;
        else
             precision[example/2] = 0.0f;

        if(trueNum != 0)
```

```
                recall[example/2] = 1.0f * correctNum / trueNum;
            else
                recall[example/2] = 0.0f;

            System.out.println("Precision  rate is : " + precision[example
            /2]);
            System.out.println("Recall rate is : " + recall[example /2]);

            correctNum = machineNum = trueNum = 0;

            for (int o=0; o<NumOutputs; o++)
            {
             //!!!!!!error += ((outData[o] - Outputs[o]) * (outData[o] -
             Outputs[o]) / 2);
              totalError += (Outputs[o] - outData[o]) * (Outputs[o] -
              outData[o]);
             //System.out.println("The " + o +"th error is : " +
             (outData[o] - Outputs[o]) * (outData[o] - Outputs[o]) / 2);
            }
        }

     float tempPrec = 0.0f;
     float tempRecall = 0.0f;

     for(int i = 0; i < data.size()/2; i++)
     {
         tempPrec += precision[i];
         tempRecall += recall[i];
     }

     tempPrec = tempPrec / (data.size()/2);
     tempRecall = tempRecall / (data.size()/2);

     System.out.println("The average precision is " + tempPrec);
     System.out.println("The average recall is " + tempRecall);
     System.out.println(" The average error is " + totalError / ( data.
     size() / 2));
 }




public void forwardPass()
{

    //System.out.println("******************BEGIN
    FORWARD********************");


    //reset the hidden layer to zero to start with
    for (int h=0; h<NumHidden; h++)
        Hidden[h] = 0.0f;

    //reset the output layer to zero to start with
    for (int o=0; o<NumOutputs; o++)
        Outputs[o] = 0.0f;

    //Get the hidden layer value
    for (int h=0; h<NumHidden; h++)
        for (int i=0; i<NumInputs; i++)
            {
                //System.out.println(" Inputs[" + i +"]: " + Inputs[i] +
                "* " + "W1[" + i +"][" + h +"]:" + W1[i][h] + "= "
                +Inputs[i] * W1[i][h]);
                Hidden[h] += Inputs[i] * W1[i][h];
            }
```

```
    //Squash the hidden nodes
    for (int h=0; h<NumHidden; h++)
         Hidden[h] = Sigmoid(Hidden[h]);

    //Get the output layer value
    for (int o=0; o<NumOutputs; o++)
    {
        for (int h=0; h<NumHidden; h++)
        {
             Outputs[o] += Hidden[h] * W2[h][o];
             //System.out.println("Output " + o + ": "+ h + "th run:" +
             Outputs[o]);
        }
    }

    //Squash the output nodes
    for (int o=0; o<NumOutputs; o++)
         {
            Outputs[o] = Sigmoid(Outputs[o]);
            //System.out.print(Outputs[o] + "   ");
         }
}


public float weightAdjust()
{
    for (int o=0; o<NumOutputs; o++)
    {
         output_errors[o] = 0.0f;
         output_errors[o] = (Outputs[o] - outData[o])* Outputs[o] * (1 -
         Outputs[o]);
         //System.out.println("Output data should be: " + outData[o] + ",
         it is "+ Outputs[o] +" error: " + output_errors[o]);
    }


    for (int h=0; h<NumHidden; h++)
    {
         hidden_errors[h] = 0.0f;

         for (int o=0; o<NumOutputs; o++)
             hidden_errors[h] += output_errors[o]*W2[h][o];

         hidden_errors[h] = hidden_errors[h]* Hidden[h] * (1 - Hidden[h]);
         //System.out.println("Hidden Error " + h +"is: " +
         hidden_errors[h]);
    }

    //Update the output weights
    for (int o=0; o<NumOutputs; o++)
    {
         for(int h=0; h<NumHidden; h++)
              {
                  //System.out.print(" W2[" + h+ "][" + o +"] from " +
                  W2[h][o] + " to ");
                  W2[h][o] = W2[h][o] + momentum * W2[h][o] - (1 -
                  momentum) * lRate * output_errors[o] * Hidden[h];
                  //System.out.println(W2[h][o]);// + " =" + momentum
                  +"(momentum)" + " - " + lRate  +" * " +
                  output_errors[o] + "*" + Hidden[h]);
              }
    }

    //Update the hidden weights
    for (int h=0; h<NumHidden; h++)
    {
          for(int i=0; i<NumInputs; i++)
```

```
                {
                    //System.out.print(" W1[" + i+ "][" + h +"] from " +
                    W1[i][h] + " to ");

         W1[i][h] = W1[i][h] + momentum * W1[i][h] - (1 - momentum) * lRate *
         hidden_errors[h] * Inputs[i];
                    //System.out.println(W1[i][h]);
                }
        }

        //Calculate the error rate
        float error = 0.0f;

        for (int o=0; o<NumOutputs; o++)
        {
            //!!!!!!error += ((outData[o] - Outputs[o]) * (outData[o] -
            Outputs[o]) / 2);
            error += (Outputs[o] - outData[o])*(Outputs[o] - outData[o]);
            //System.out.println("The " + o +"th error is : " + (outData[o] -
            Outputs[o]) * (outData[o] - Outputs[o]) / 2);
        }
        return error;
    }


    public void changeRate(float rate)
    {
        lRate = rate;
    }


    public void print()
    {
        System.out.println(" The input layer has " + NumInputs + " nodes");
        System.out.println(" The hidden layer has " + NumHidden + " nodes");
        System.out.println(" The output layer has " + NumOutputs + " nodes");
        System.out.println(" The threshold is " + threshold);
        System.out.println(" Momentum: " + momentum + " Learning Rate: " +
        lRate);

        System.out.println("The weight matrix:");
        for (int ii=0; ii<NumInputs; ii++)
                for (int hh=0; hh<NumHidden; hh++)
                    System.out.print(W1[ii][hh]+ "   ");

        System.out.println();

        for (int hh=0; hh<NumHidden; hh++)
                for (int oo=0; oo<NumOutputs; oo++)
                    System.out.print(W2[hh][oo]+"    ");


    }

    protected float Sigmoid(float x)
    {
        return (float)(1.0f/(1.0f+Math.exp((double)(-x))));
    }
    protected float[] hidden_errors;
}
```

```
import java.util.*;

class INN extends Neural

{
   public Vector transformedData;    //data after transformation for the
   purpose of INN
   public int numNewNodes;           //Keep track of incremented classes
   number


   INN(String NNfile_name, String dataFile_name, float thresh,float lRate,
   float momentum, int numNewNodes)
   {

     this.numNewNodes = numNewNodes;
     transformedData = new Vector();

     neuralFile = new NNfile(NNfile_name);
     NumInputs = neuralFile.numInput;
     NumHidden = neuralFile.numHidden;
     NumOutputs = neuralFile.numOutput + numNewNodes;
     WeightsFlag= neuralFile.weightFlag;

     Inputs = new float[NumInputs];
     Hidden = new float[NumHidden];
     Outputs = new float[NumOutputs];

     W1 = new float[NumInputs][NumHidden];
     W2 = new float[NumHidden][NumOutputs];

     //Set up all the training parameters
     threshold = thresh;
     this.lRate = lRate;
     this.momentum = momentum;

     // Retrieve the weight values from the NNfile object:
     if (WeightsFlag)
     {
       for (int i=0; i<NumInputs; i++)
            for (int h=0; h<NumHidden; h++)
                W1[i][h] = neuralFile.getW1(i, h);

       for (int h=0; h<NumHidden; h++)
          for (int o=0; o<NumOutputs; o++)
            W2[h][o] = 0.0f;

     }

     else
       randomizeWeights();

     //System.out.println("This is after saving: " +  + this.W1[20][14] +"  "
     + this.W1[15][20] + "  " + this.W2[20][14] +"  " + this.W2[15][20] + "
     ");

     //Get the training or testing data
     trainSet = new DataFile(dataFile_name);
     data = trainSet.getData();

     //print();
   }


   public void randomizeWeights()
   {
      // Randomize weights here:
```

```
    // 64 inputs nodes with value of 0 or 1, the summed weight aims at [-1,
    1] so that the
    // hidden nodes will be about 0.5
    for (int ii=0; ii<NumInputs; ii++)
       for (int hh=0; hh<NumHidden; hh++)
            W1[ii][hh] =  0.01f * (float)Math.random();

    for (int hh=0; hh<NumHidden; hh++)
       for (int oo=0; oo<NumOutputs; oo++)
            //W2[hh][oo] = (float)Math.random() - 0.5f;
            {
                if(Math.random() >= 0.5)
                  W2[hh][oo] = 1;
                else
                  W2[hh][oo] = -1;
            }
}


public void transformData()
{

   for(int example = 0; example < data.size(); example+=2 )
    {

        //System.out.println("example number " + example);

        // copy the input values:
        for (int i=0; i<NumInputs; i++)
               Inputs[i] = ((int[])data.elementAt(example))[i] ;


        //copy the training output
        outData = new int[NumOutputs];

        for (int i=0; i< NumOutputs; i++)
               outData[i] =((int[])data.elementAt(example+1))[i];

        setHiddenNodes();
        transformedData.addElement(outData);
     }
}



public void setHiddenNodes()
{

    Hidden = new float[NumHidden];

    for (int h=0; h<NumHidden; h++)
        Hidden[h] = 0.0f;

    //Get the hidden layer value
    for (int h=0; h<NumHidden; h++)
        for (int i=0; i<NumInputs; i++)
                  Hidden[h] += Inputs[i] * W1[i][h];

    //Squash the hidden nodes
    for (int h=0; h<NumHidden; h++)
         Hidden[h] = Sigmoid(Hidden[h]);

    transformedData.addElement(Hidden);
}


public void trainSecondNN()
```

```
    {
        TwoLayerNN secondNN = new TwoLayerNN( NumHidden, NumOutputs,
        transformedData, threshold, lRate, momentum);

        float error[] = new float[5000];

        for(int i=0;;i++)
                {

                    error[i] = secondNN.train();
                    System.out.println(" The " + i +"th turn: " + error[i]);
                    if( error[i] < 0.1 || i==4999)
                    {
                        System.out.println("The error is " + error[i]);
                        int totalEpoch = i;
                        System.out.println("It takes "+totalEpoch + "
                        epochs");
                        break;
                    }
                }

        W2 = secondNN.getW();
    }

    public static void main( String[] args)
    {
        INN test = new INN("Z-1nn", "ZTrain03.txt",0.5f, 0.5f, 0, 1);
        test.transformData();

        Date startTime = new Date( );
        test.trainSecondNN();
        //NNfile.saveINN("NN05_03.txt",test);
        Date stopTime = new Date( );

        long elapsedTime = stopTime.getTime( ) - startTime.getTime( );
        System.out.println( "elapsed time is " + elapsedTime );

        test.test("ZTest03.txt");
    }
}
```

```
import java.util.*;

public class TwoLayerNN
{
        /**********************Data about neural net
        characteristics*********************/

        protected int NumInputs;
        protected int NumOutputs;

        protected boolean WeightsFlag;
        protected float[][] W;

        public float[] Inputs;
        public float[] Outputs;
        public int[] outData;       //The output in training data

        protected float[] output_errors;

        protected float[] totalError;

        protected Vector data;

        public NNfile neuralFile=null;
        public DataFile trainSet=null;


        /************************Data about training
        parameters*****************************/

        public float threshold;  //Used for testing
        public float lRate;      //learning rate
        public float momentum;   //momentum

        TwoLayerNN()
        {
          NumInputs = NumOutputs = 0;
        }


        TwoLayerNN(int input, int output, Vector dataFile, float thresh,float
        lRate, float momentum)
        {
          // Get  a file  and construct a net

          /********** First Step: Construct the framework of the neural net
          **************/

          NumInputs = input;
          NumOutputs = output;

          Inputs = new float[NumInputs];
          Outputs = new float[NumOutputs];
          outData = new int[NumOutputs];

          W = new float[NumInputs][NumOutputs];
          randomizeWeights();


          //Set up all the training parameters
          threshold = thresh;
          this.lRate = lRate;
          this.momentum = momentum;

          //Get the training or testing data
          data = new Vector();
          data = dataFile;
```

```
  output_errors = new float[NumOutputs];
  totalError = new float[data.size()/2];
  System.out.println("The errors are" + data.size()/2);
}



public void randomizeWeights()
{
   // Randomize weights here:
   for (int ii=0; ii<NumInputs; ii++)
      for (int hh=0; hh<NumOutputs; hh++)
           W[ii][hh] =  0.01f * (float)Math.random();
}


public float train()
{

    //System.out.println("Learning rate is " + lRate);
    for (int example = 0; example < data.size(); example+=2 )
    {
        // zero out error arrays:
        for (int o=0; o<NumOutputs; o++)
             output_errors[o] = 0.0f;

        // copy the input values:
        for (int i=0; i<NumInputs; i++)
               Inputs[i] = ((float[])data.elementAt(example))[i];

        //copy the training output
        for (int i=0; i<NumOutputs; i++)
               outData[i] =((int[])data.elementAt(example+1))[i];

        //System.out.println();

        forwardPass();

        totalError[example / 2] = weightAdjust();
     }

    float error = 0.0f;

    for(int i = 0; i < data.size() / 2; i++)
          error += totalError[i];

    error /= (data.size()/2);
    return error;
}


public void forwardPass()
{

    //reset the output layer to zero to start with
    for (int o=0; o<NumOutputs; o++)
        Outputs[o] = 0.0f;

    //Get the output layer value
    for (int o=0; o<NumOutputs; o++)
        for (int i=0; i<NumInputs; i++)
             Outputs[o] += Inputs[i] * W[i][o];

    //Squash the output nodes
    for (int o=0; o<NumOutputs; o++)
            Outputs[o] = Sigmoid(Outputs[o]);
}
```

```
public float weightAdjust()
{
    for (int o=0; o<NumOutputs; o++)
    {
         output_errors[o] = 0.0f;
         output_errors[o] = (Outputs[o] - outData[o])* Outputs[o] * (1
         - Outputs[o]);
    }

    //Update the output weights
    for (int o=0; o<NumOutputs; o++)
         for(int i=0; i<NumInputs; i++)
                  W[i][o] = W[i][o] + momentum * W[i][o] - (1 -
                  momentum) * lRate * output_errors[o] * Inputs[i];


    //Calculate the error rate
    float error = 0.0f;

    for (int o=0; o<NumOutputs; o++)
        error += (Outputs[o] - outData[o])*(Outputs[o] - outData[o]);

    return error;
}



 public void test(Vector dataFile)
 {

     int result[] = new int[NumOutputs];
     float[] precision = new float[data.size()/2];
     float[] recall = new float[data.size()/2];
     float totalError  = 0.0f;
     int correctNum = 0;
     int trueNum = 0, machineNum = 0;

     data = dataFile;

     for (int example = 0; example < data.size(); example+=2 )
     {

        // copy the input values:
        for (int i=0; i<NumInputs; i++)
        {
           Inputs[i] = ((float[])data.elementAt(example))[i];
           //System.out.print(Inputs[i]);
        }

        //System.out.println();

        //copy the training output
        for (int i=0; i<NumOutputs; i++)
            {
                outData[i] =((int[])data.elementAt(example+1))[i];
                //System.out.print(outData[i]);
            }

        //System.out.println();

        forwardPass();

        for(int i =0; i<NumOutputs; i++)
             System.out.print(outData[i]);
        System.out.println();
```

```
            for(int i =0; i<NumOutputs; i++)
            {
                 //System.out.print(Outputs[i] + "-->");
                 if(Outputs[i] >= threshold)
                   result[i] = 1;
                 else
                   result[i] = 0;
                 System.out.print(result[i]);
            }
            System.out.println();
            System.out.println();

            //Calculate the accuracy
            for(int i =0; i<NumOutputs; i++)
            {
                if(outData[i] == result[i] && result[i]==1)
                {
                    correctNum++;
                    machineNum++;
                    trueNum++;
                }

                else
                {
                    if(result[i] == 1)
                      machineNum++;

                    if(outData[i]==1)
                      trueNum++;
                }
            }

            precision[example/2] = 1.0f * correctNum / machineNum;
            recall[example/2] = 1.0f * correctNum / trueNum;
            System.out.println("Precision  rate is : " + precision[example
            /2]);
            System.out.println("Recall rate is : " + recall[example /2]);

            correctNum = machineNum = trueNum = 0;

            for (int o=0; o<NumOutputs; o++)
            {
             //!!!!!!error += ((outData[o] - Outputs[o]) * (outData[o] -
             Outputs[o]) / 2);
              totalError += (Outputs[o] - outData[o]) * (Outputs[o] -
              outData[o]);
             //System.out.println("The " + o +"th error is : " +
             (outData[o] - Outputs[o]) * (outData[o] - Outputs[o]) / 2);
            }
        }

    float tempPrec = 0.0f;
    float tempRecall = 0.0f;

    for(int i = 0; i < data.size()/2; i++)
    {
        tempPrec += precision[i];
        tempRecall += recall[i];
    }

    tempPrec = tempPrec / (data.size()/2);
    tempRecall = tempRecall / (data.size()/2);

    System.out.println("The average precision is " + tempPrec);
    System.out.println("The average recall is " + tempRecall);
    System.out.println(" The average error is " + totalError / ( data.
    size() / 2));
}
```

```
    public void changeRate(float rate)
    {
        lRate = rate;
    }

    protected float Sigmoid(float x)
    {
        return (float)(1.0f/(1.0f+Math.exp((double)(-x))));
    }

    public float[][] getW()
    {
        return W;
    }
}
```

```
import java.util.*;

public class TestNeural
{
    public boolean validation;
    public boolean doEpoch;
    public int maxEpoch;
    public float maxError;
    public float threshold;

    public float lRate;
    public float momentum;

    private Neural myNeural;
    private float[] error;
    private int totalEpoch;


    public TestNeural(String NNfile_name, String dataFile_name, boolean epo,
    int epoNum, float error, float thresh,float lRate, float momentum)
    {
        //Initiate the parameters for training

        doEpoch = epo;
        maxEpoch = epoNum;
        maxError = error;
        threshold = thresh;
        this.lRate = lRate;
        this.momentum = momentum;

        myNeural = new Neural(NNfile_name, dataFile_name, thresh, lRate,
        momentum);
        if(doEpoch)
            this.error = new float[maxEpoch];
        else
            this.error = new float[100000];
    }


    public void train()
    {
        if(doEpoch)
            {
                float tempError = 0.0f;
                Date startTime = new Date( );

                for(int i = 0; i < maxEpoch; i++)
                {

                    if( i > 32)
                        myNeural.changeRate((float)(lRate / ( 1 + i / 32)));

                    error[i] = myNeural.train();
                    //error[i] = myAda.train();
                    tempError += error[i];
                    //System.out.println( "The " + i +"th turn: " +
                    error[i]);
                    //System.out.println("ATTENTION!!" + myNeural.W1[1][2] +
                    "    " + myNeural.W2[3][2]);
                }

                System.out.println("The average error is: " + tempError /
                maxEpoch);

            }
        else
            for(int i=0;;i++)
            {
```

```
                //if( i > 32)
                //  myNeural.changeRate((float)(lRate / ( 1 + i / 32)));

                error[i] = myNeural.train();
                //error[i] = myAda.train();
                System.out.println(" The " + i +"th turn: " + error[i]);
                if( error[i] < maxError || i==(maxEpoch - 1))
                //if(   error[i] < maxError)
                {
                    System.out.println("The error is " + error[i]);
                    totalEpoch = i;
                    System.out.println("It takes "+totalEpoch + " epochs");
                    break;
                }
            }

        //System.out.println("This is before saving: " + myNeural.W1[20][14]
        +"  " + myNeural.W1[15][20] + "  " + myNeural.W2[20][14] +"  " +
        myNeural.W2[15][20] + "  ");

       NNfile.saveFile("ZNN1.txt", myNeural);

    }

    public void test(String testFile)
    {
        myNeural.test(testFile);
        //myAda.test(testFile);

        //System.out.println("AND NOW>>>" + myNeural.W1[1][2] + "   " +
        myNeural.W2[3][2]);
    }



    public void printNet()
    {
        myNeural.print();
        System.out.println(" We use epoches? " + doEpoch);
    }




    public static void main(String[] args)
    {
        TestNeural x = new TestNeural( "test7.txt", "ZTrain02.txt", false,
        5000, (float) 0.1,  (float) 0.5, 0.5f, 0.0f);
        x.printNet();

        Date startTime = new Date( );
        x.train();
        Date stopTime = new Date( );
        long elapsedTime = stopTime.getTime( ) - startTime.getTime( );
        System.out.println( "elapsed time is " + elapsedTime );

        x.test("ZTest02.txt");
    }

}
```

```
/**
  Class TextReader provides methods for reading character type data an input
  source, either the keyboard or a text file (characters such as digits and
  letters only).  TextReader can be constructed either from an InputStream
  such as System.in (the keyboard) and in fact is done automatically with

    TextReader keyboard = new TextReader( );

  or by specifying the name of a file as a String as in

    TextReader inputFile = new TextReader( "input.data" );

  Written by Stuart Reges at the Univerity of Arizona 6/11/98 with
  minor modifications by Rick Mercer.
*/
import java.io.*;

public class TextReader
{
//--instance variables
  // PushbackReader used here to avoid bugs in the 1.1 BufferedReader class
  // It also allows for a "peek" method.
  private PushbackReader in;   // the input stream
  // true If from keyboard or false when input is from a file
  private boolean rePrompting; // users should be prompted, but not files

/**
  Construct an object used to obtain input from the keyboard
  */
  public TextReader(  )
  { // pre : input stream is open for reading
    // post: constructs a TextReader object associated with the keyboard
    in = new PushbackReader( new InputStreamReader( System.in ) );
    rePrompting = true;
  }

/**
  Construct an object used to obtain input from the disk file with
  only text data such as letters, digits and other characters like * % $
  */
  public TextReader( String fileName )
  { // pre : fileName is the name of a file that can be opened for reading
    // post: constructs a TextReader tied to the given file
    try {
      in = new PushbackReader(new FileReader(fileName));
      rePrompting = false;
    }
    catch( Exception e ) {
      System.out.println("Can't open input file '" + fileName +
      "', program terminated");
      System.exit(1);
    }
  }

  private void error( String where )
  { // Have a standard way of displaying error messages
    System.out.println("\n***Failure in " + where +
    " message. Program terminated***" );
    System.exit( 1 );
  }

/**
  Allows user to read in a True or fAlSe value into a boolean variable.
  The case on input does not matter, but only the strings TRUE or FALSE are
  accepted
  */
  public boolean readBoolean()
  {  // return true
```

```
    do
    {
      String torf = readWord( );
      if( torf.equalsIgnoreCase( "true" ) )
        return true;
      else if( torf.equalsIgnoreCase( "false" ) )
        return false;
      else
       System.out.println( torf + " is not 'true' or 'false'. Try again");
    } while( true );
  }

/**
  Use this method to read in entire lines of data such as
  a persons full name or an address.  You can also use it to enter
  a string that maay or may not have blanks spaces.
  Precondition: The input is not at end-of-file of the input stream.
  @return All characters (including blank spaces) up until the enter
  key is entered.
  */
  public String readLine( )
  {
    String result = "";
    try {
      do
      {
        int next = in.read( );
        if( next == '\r' ) // skip carriage-return on Windows systems
          continue;
        if( next == -1 || next == '\n' )
          break;
        result += (char)next;
      } while(true);
    }
    catch( Exception e ) {
      error ( "readLine" );
    }
    return result;
  }

/**
  Returns the next character on the input stream. Same as read.
  Precondition: The input is not at end-of-file of the input stream.
  @return The next character in the input stream.
  */
  public char readChar()
  {
    return read( );
  }

/**
  Returns the next character on the input stream. Same as readChar.
  Precondition: The input is not at end-of-file of the input stream.
  @return The next character in the input stream.
  */
  public char read( )
  { // pre : not at end-of-file of the input stream
    // post: reads the next character of input and returns it

    char result = ' ';
    try {
      result = (char)in.read();
      if (result == '\r') // skip carriage-return on Windows systems
        result = (char)in.read();
    }
    catch( Exception e ) {
      System.out.println(
      "Failure in call on read method, program terminated." );
```

```
      System.exit( 1 );
    }
    return result;
  }

/**
  Puts the given character back into the input stream to be read again
  */
  public void unread( char ch )
  {
    try {
      in.unread((byte)ch);
    }
    catch( Exception e )
    {
      error( "unread" );
    }
  }

/**
  Use this to find out what the next character is when what you do depends
  on it. Especially useful when processing complex file input with unknown
  amounts of input data. The peek method allows for multiple sentinels
  @return the next character in the input stream without actually reading it.
  */
  public char peek( )
  {
    int next = 0;
    try {
      next = in.read( );
    }
    catch( Exception e ) {
      error( "peek" );
    }
    if( next != -1 )
      unread( (char)next );
    return (char)next;
  }

/**
  Reads one string (terminated by end-of-file or whitespace). This method
  will skip any leading whitespace.
  Precondition stream contains at least one nonwhitespace character
  @returns The next string in the input disk file or
  that is typed at the keyboard.
  */
  public String readWord()
  {
    String result = "";
    try {
      int next;
      do
      {
        next = in.read();
      } while( next != -1 && Character.isWhitespace( (char)next) );

      while (next != -1 && !Character.isWhitespace( (char)next ) )
      {
        result += (char)next;
        next = in.read();
      }

      while (next != -1 && next != '\n' &&
      Character.isWhitespace((char)next))
      {
        next = in.read();
      }
```

```
      if (next != -1 && next != '\n')
        unread((char)next);
      } // end try
      catch (Exception e)
      {
        error( "readWord" );
      } // end catch

    return result;
  }

/**
   Reads an int and skips any trailing whitespace on current line.
   Keeps trying if floating point number is invalid.
   Precondition: next token in input stream is an int.
   @returns The next integer in the input disk file or that is typed
   at the keyboard.
  */
  public int readInt()
  {
    int result = 0;
    do // keep on trying until a valid double is entered
    {
      try
      {
        result = Integer.parseInt(readWord());
        break;  // result is good, jump out of loop down to return result;
      }
      catch (Exception e)
      {
        if(rePrompting)
          System.out.println("Invalid integer. Try again.");
        else
        {
          error( "readInt" );
          break;
        }
      }
    } while( true );
    return result;
  }

/**
   Reads an floating point numnber and skips any trailing whitespace on
   current line. Keeps trying if floating point number is invalid.
   Precondition: next token in input stream is an valid number (int or
   double).
   @returns The next floating point number in the input disk file or
   that is typed at the keyboard.
  */
  public double readDouble()
  { // pre : next token in input stream is a double
    // post: reads double and skips any trailing whitespace on current line
    //       Keeps trying if floating point number is invalid
    double result = 0.0;

    do  // keep on trying until a valid double is entered
    {
      try {
        result = new Double(readWord()).doubleValue();
        break;  // result is good, jump out of loop down to return result;
      }
      catch( Exception e )
      {
        if(rePrompting)
          System.out.println("Invalid floating-point number. Try again.");
        else
        {
```

```
            error("readDouble");
            break;
          }
        }
      } while( true );
      return result;
    }

  /**
    Find out if the input stream has more data that can be read.
    This is especially useful when processing file data where the
    size of the file is not determined (or changes a lot).
    @returns true if input stream is ready for reading otherwise returns false
    */
    public boolean ready( )
    {
      boolean result = false;
      try {
        result = in.ready();
      }
      catch (IOException e) {
        error ( "ready" );
      }
      return result;
    }
  }
```